

(19)



Europäisches Patentamt
European Patent Office
Office européen des brevets



(11)

EP 0 813 129 A2

(12)

EUROPEAN PATENT APPLICATION

(43) Date of publication:

17.12.1997 Bulletin 1997/51

(51) Int Cl.⁶: G05B 19/042, G05B 19/418

(21) Application number: 97304138.7

(22) Date of filing: 13.06.1997

(84) Designated Contracting States:

AT BE CH DE DK ES FI FR GB GR IE IT LI LU MC
NL PT SE

(30) Priority: 14.06.1996 US 665211

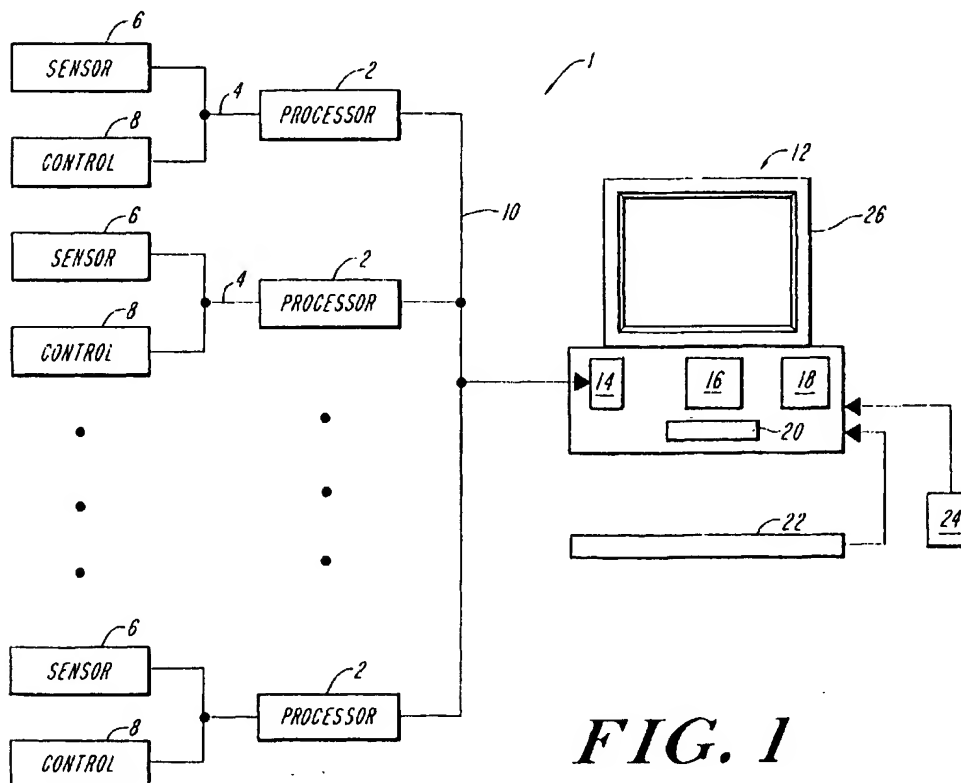
(71) Applicant: **THE FOXBORO COMPANY**
Foxboro, MA 02035 (US)(72) Inventor: **Calder, Dale E.**

Mansfield, Massachusetts 02048 (US)

(74) Representative: **Downing, Michael Philip et al**
Fry Heath & Spence,
The Old College,
53 High Street
Horley, Surrey RH6 7BN (GB)**(54) Keypad annunciator graphical user interface**

(57) A Keypad Annunciator Graphical User Interface, (KAGUI), for use by an operator of a process control system for presenting and responding to alarm state data generated by the process control system, is disclosed. The KAGUI provides an interactive display of a keypad annunciator on a monitor display screen. A pan-

el manager display and a dynamic icon enhance operator awareness, in a multi-window display environment, of alarms. The KAGUI synchronizes alarm data presented by the KAGUI with process control system alarm data and informs the operator of the status of interprocess communication.

**FIG. 1**

EP 0 813 129 A2

Description

The present invention relates to devices for monitoring and controlling industrial processes, and more particularly, to a keypad annunciator graphical user interface for presenting process control system alarm data, and for providing a flexible and intuitive means by which an operator may acknowledge alarms and take corrective action.

Modern industrial processes are becoming increasingly automated and complex. Such processes, whether involving the smelting of steel or the generation of nuclear power, are typically overseen from a centralized location by a process engineer or operator, who monitors data from hundreds, or even thousands, of sensors distributed throughout a facility. Sensors measuring voltage, current, power consumption, magnetic or electric field strength, temperature, pressure, humidity -- the list is virtually endless -- can be part of a modern process control system. Sensor data is used to evaluate the suitability of raw materials, provide feedback data during processing and manufacture, control valves and process flows, ensure the quality of manufactured products, and generate alarms. In particular, sensors can be used to gather data indicative of an alarm state in a process control system.

When an alarm occurs, considerable demands are placed on the process operator, who, confronted with a vast array of data from a vast array of sensors, must respond quickly and effectively to ensure safe and economical operation of the facility. Optimally, when an alarm occurs the operator is immediately alerted. Next, the operator ascertains the location, purpose and significance of the sensor. Typically, there are thousands of pages of documentation describing the operation of the industrial process in question and describing the purpose and significance of the various sensors. After locating and consulting the appropriate portions of the documentation, and perhaps viewing data from other sensors, the operator institutes proper corrective action. The actual scenario, however, is not always optimal, and the operator's job is at times difficult.

In an earlier era, of simpler, less automated industrial processes, an annunciator wall panel often informed operators of alarms. Individual sensors were essentially hard wired, via relays or other simple circuitry, to individual lamps placed at appropriate locations on a wall-sized schematic of the process. Sensor data out of an acceptable range illuminated a light and sounded a horn. However, technological advances, particularly the advent of computers, have greatly increased the complexity of most industrial processes, as well as increased the burden on the operator charged with oversight of the process. In many cases, the old annunciator panel wall is obsolete and relatively inflexible.

Technological advance has also improved the tools used by the process operator. Video displays run application programs that can, for example, graph process

data, display a process graphic (a schematic representation of a portion or of all of an industrial process) or perform a keyword search through documentation. However, not all the tools are significantly improved: the hardware annunciator keyboard (HAK) remains. The HAK is composed of a numeric keypad with adjacent rows of illuminable lights, and can be described as essentially a shrunk version of the annunciator wall panel. An occurrence of a sensor alarm causes a horn to sound and illuminates a key associated with that alarm. The operator acknowledges the alarm and silences the horn by pressing the illuminated key.

The present HAK does not optimally address the needs of an operator charged with overseeing a modern and highly automated modern industrial process. The existing HAK is bulky, costly to produce, difficult to reconfigure, and limited to a number of standard key formats. The HAK warns an operator of all alarms, regardless of severity, in the same manner -- the same horn sounds, and one of several keys, all typically of the same color, illuminates. Adding alarm states to a process control system can require multiple hardware annunciator keyboards, as keys are not easily added to an existing keyboard, further adding to system costs.

Accordingly, a need exists for a more economical and versatile alarm interface that relatively quickly and effectively communicates alarm states, facilitates timely and proper corrective action, and complements the graphical tools for video display already at the operator's disposal.

Embodiments of the invention provide a more versatile keypad annunciator that allows an operator to manage more effectively and efficiently a modern complex industrial process control system; and that more effectively presents alarm data at lower cost.

The preferred embodiments of the invention also provide an operator-configurable keypad annunciator that an operator may readily configure to better ensure acknowledgment and appropriate operator response to the occurrence of an alarm state.

The invention is defined in the claims appended hereto. However, the following discussion may be of assistance.

Embodiments of the invention meet the above needs by providing a keypad annunciator graphical user interface for displaying an interactive keypad annunciator on a monitor display screen. The keypad annunciator display is interactive, i.e., a user may enter commands and otherwise direct the process control system by selecting elements of the keypad display, such as a key, with a pointing and selecting input device, such as a mouse. The keypad annunciator graphical user interface, or KAGUI, is also readily configurable by a user such that presentation of alarm data and of actions initiated by selection of the key may be tailored to better manage the process control system. The KAGUI complements other window-based tools at the operator's disposal, allowing use of other windows, yet providing

an enhanced ability to alert an operator of an occurrence of an alarm state. The term "alarm state," as used herein, refers to a state of the process control system determined as a function of system inputs that include sensor data. A normal alarm state is one in which no alarms are occurring (i.e., sensor data and other inputs indicate that the process is operating in a normal or acceptable manner). In comparison, when there is a potential problem with the process, an alarm is generated in order to quickly notify the operator, and a particular alarm state is said to have occurred.

According to one aspect of the present invention, a key of the KAGUI is associated with an alarm state or states of the process control system such that occurrence of the alarm state causes the key to change its appearance. Typically, the key changes color and blinks. The invention also provides for a system wherein the operator chooses a particular key to associate with a particular alarm state and defines the alarm states.

Further aspects of the invention provide for user interaction. In particular, the system provides an input element that is selectable and acknowledges a key in an alarm state by selecting the key with a pointing device. Selection of a key typically silences the horn and displays a process graphic, i.e., a schematic representation of the area of the facility where the alarm is occurring. Such a process graphic can include summary data on the sensor or sensors generating the alarm.

The KAGUI is not likely to be the only application using the display monitor of a process control system, and at times other windows and GUIs are likely to obscure the KAGUI. These other windows and GUIs can create confusion -- a user can open several windows simultaneously, perhaps temporarily losing track of other important windows. Alarm data are too important to be ignored, however, even temporarily. Accordingly, to provide the advantages of a KAGUI operating in a modern application environment, yet ensure proper presentation of alarm data, the invention incorporates several features.

According to one feature of the present invention, occurrence of an alarm state associated with a key of a particular keypad automatically causes that keypad to appear as the topmost window in the display.

According to another feature of the present invention, a user may choose an option whereby a keypad annunciator floats on top of the display as a high priority window, i.e., a window with a priority higher than a majority of other windows.

According to yet another feature, the invention provides a dynamic icon, representing a minimized keypad annunciator, that changes appearance to indicate occurrence of any alarm state associated with selected keys. The icon normally appears in a static state; however, upon occurrence of an alarm state associated with the selected keys, the icon appears in a dynamic state, exhibiting a selected motion. The icon can also simply change from one static display state, indicating no

alarms, to a second static display state, indicating an alarm. The icon may be configured to appear on the display upon the occurrence of an alarm state, or to float on the display as a high priority icon.

According to yet a further feature of the present invention, the KAGUI incorporates an interactive compact visual representation of alarm data. This compact visual representation or display of alarm state information is referred to herein as a panel manager. The panel manager is typically larger than an icon but smaller than a full keypad annunciator display.

In one aspect of the present invention, the panel manager comprises unique graphical elements, such as buttons, that are each associated with a keyboard annunciator. The graphical element changes visual appearance, e.g., changes color and/or blinks, upon occurrence of an alarm state associated with a selected key or keys of the particular keypad annunciator represented by the button. Selection of the button with a pointing device, such as a pen or mouse, causes the appropriate keypad annunciator to appear on the display. As usual, the alarm state occurrences are accompanied by the sounding of a horn or generation of whatever audible stimulus is associated with that alarm state by the operator.

In another aspect of the present invention, the buttons of a panel manager employ a selectable third visual state to indicate existence of an alarm state that has been acknowledged by an operator.

In a further aspect of the present invention, the panel manager comes to the front as a topmost window upon the occurrence of an alarm state associated with a key on a keypad of which the panel manager is a compact visual representation. The panel manager may also simply be made a display with a higher priority than most other windows of the display. Because of its compact size, placing the panel manager at the forefront of the display does not unduly interfere with the operator's working with other windows.

The foregoing aspect of the invention regarding presentation of alarm states, particularly the dynamic icon and the panel manager, allows an operator to fully employ a multi-window, multi-application control workstation yet be highly assured of timely and effective communication of the occurrence of an alarm state.

The keyboard annunciator graphical user interface communicates to an operator the alarm status of a process control system. A typical industrial process control system is highly distributed and typically uses computer processors of different manufacture, often running different operating systems, located at a distance from each other. Communication between computers is often over a network. Interprocess communication (IPC) is therefore the lifeblood of the modern and highly automated process control system. Sensors can communicate with processors, processors can communicate with each other, and many of the foregoing can communicate with the control workstation. Network communication,

however, is not always reliable, and alarm data transmission to the KAGUI can be interrupted. Nevertheless, the KAGUI should accurately represent at all times the alarm state of the process control system.

In a healthy industrial process, alarm states do not regularly occur and the communication means for transmitting alarm data are not constantly in use and thus are not constantly tested. Accordingly, the invention provides apparatus and methods to maximize probability that the alarm state of the process control system represented by the keyboard annunciator graphical user interface is correct.

According to one feature of the present invention, "verification" messages are constantly transmitted over a communication path to verify its continued existence. Acknowledgment signals are sent by a receiver of an IPC message to the sender. Under normal conditions, that is, when the communication path is functional properly and open, constantly sending verification signals results in constantly receiving acknowledgment signals, which confirm the continued existence of the path. The process operator is typically informed of the status of the communication path by a display element of the KAGUI. The display element typically has two visual states, one to indicate successful transmission of verification signals and another to indicate unsuccessful transmission of verification signals, and hence a break in the communication channel. Note that the content of the verification message is typically irrelevant. It is the sending of the messages on a regular basis that serves to verify the continued existence of the IPC path.

According to another aspect of the present invention, means are provided, such as a selectable button on a toolbar, for an operator to resynchronize the alarm state information possessed and displayed by a keyboard annunciator graphical user interface with that of the process control system. Resynchronization refers to a download of alarm state information from a process control system to a KAGUI to refresh the data presented by the KAGUI so that the KAGUI represents, as accurately as possible, the alarm state of the process control system.

In yet a further aspect of the present invention, the KAGUI automatically maintains synchronization of alarm data by, upon detecting a failed transmission "verification signal," reestablishing the communication path, such as a pipe, between the KAGUI and the process control system and retransmitting all alarm state information from the process control system to the KAGUI.

In an additional aspect of the present invention, a server process is established to route alarm state data from a state table maintained by a process control system to a client process operative with the KAGUI. Such a server process can serve multiple clients.

The KAGUI can be configured in many ways to enhance presentation of, and response to, alarm state information. Configuration options useful to an operator are therefore briefly outlined.

According to another aspect of the present invention, the KAGUI appears on a monitor as a traditional hardware keyboard annunciator. A numeric keypad is situated adjacent rows of alarm keys.

In an additional aspect of the present invention, the layout of the keys, that is, the number of rows and columns, as well as the overall number of keys of the keypad annunciator are selectable. Multiple keypad annunciators are also possible. Multiple lines of label information are displayable on the surface of each of the keys of the keypad annunciator. Furthermore, the visual appearance of a key for indicating non-occurrence of a alarm state, occurrence of an alarm state, and an acknowledged alarm are selectable. Typically, a key not in alarm appears as a first solid color; a key in alarm appears as a second color and blinking; and a key indicating an acknowledged alarm appears in the second color but does not blink.

An audible stimulus typically alerts the operator to a key in an alarm state. However, all alarm states are not created equal; some may involve potential injury to personnel while others may indicate a simple and relatively easily correctable "glitch" in the industrial process. Accordingly, the invention allows an operator to select a unique sound to indicate priority of an alarm.

Embodiments of the invention will now be described by way of example, with reference to the accompanying drawings, in which;

FIGURE 1 is a schematic illustration of a typical process control system;

FIGURE 2 illustrates selected software and hardware components of process control system including a keypad annunciator graphical user interface;

FIGURE 3 is a typical process graphic, displayable on a workstation monitor, of a process of a process control system;

FIGURE 4 illustrates a keypad annunciator graphical user interface and a process graphic displayed alongside each other on a workstation monitor display;

FIGURES 5A and 5B illustrate user-configured keypad annunciators of a keypad annunciator graphical user interface;

FIGURE 6 shows a typical panel manager display component of the keypad annunciator graphical user interface;

FIGURE 7 illustrates the panel manager displayed alongside a process graphic on a workstation display monitor;

FIGURES 8A and 8B show a dynamic icon for displaying a minimized panel manager component of a keypad annunciator graphical user interface;

FIGURE 9 illustrates a property page for configuring the major display options of the keypad annunciator graphical user interface;

FIGURE 10 is a flow chart illustrating run-time decisions made by the keypad annunciator graphical

user interface for displaying alarm state information; and

FIGURE 11 is another flow chart of the run time behavior of the keypad annunciator graphical user interface, illustrating the maintenance of interprocess communication paths used for transmission of alarm data, including automatic synchronization of process control system alarm data and alarm state information presented by a keypad annunciator graphical user interface.

FIGURE 1 schematically illustrates the major components of a modern process control system 1. Processors, such as processor 2, are typically located throughout an industrial facility and serve to process data from sensors, such as sensor 6, and to operate controls, such as control 8. Sensor 6 provides feedback to processor 2 such that programs or algorithms running on the processor 2 can properly operate control 8. For example, sensor 2 could be a temperature sensor for measuring the temperature of a process flow and control 6 could be a valve for controlling the flow. Processor 2 also calculates alarm states based on data from sensor 6, and perhaps from other sensors as well.

Processor 2 can be a complete computer, including a central processing unit, random access memory, read only memory, a hard drive and input-output hardware. A monitor is usually not part of processor 2, nor is a keyboard typically attached. Several different processors may be part of the overall process control system 1, and different operating systems such as UNIX, SUN OS or Windows NT, running on different machines, can be part of an overall process control system. Processor 2 communicates with sensor 6 and with control 8, and with the central control and monitoring workstation 12, over interprocess communication (IPC) paths 4 and 10.

An operator uses workstation 12 to monitor and control the industrial process. Workstation 12 typically comprises input-output hardware 14, a central processor 16, random access memory 18, disk storage 20, a hardware annunciator keyboard (HAK) 22, pointing device, such as mouse 24 and a display monitor 26. A regular keyboard (not shown) for the entry of text is typically also included. The invention replaces HAK 22 with a keypad annunciator graphical user interface (KAGUI). The KAGUI is displayed on video screen 26.

In a simple process control system, individual processors, such as processor 2, are not used, and workstation 12 operates controls, such as control 8, communicates with sensors, such as sensor 6, and runs the computational tasks associated with the process control system 1, including the KAGUI.

FIGURE 2 illustrates several software components important to the invention. Alarm alert task 30, alarm server task 42, and display managers 46 are part of a typical process control system. Server task 34 and client task 38 work closely with the KAGUI. Communication between tasks, and between tasks and hardware com-

ponents such as hardware annunciator keyboard 50 and workstation monitor 26, is accomplished via IPC mechanisms. For example, alarm alert task 30 communicates with the server task 34 using IPC path 32. Interprocess communication is important to the present invention and to a process control system, and techniques regarding IPC are subsequently discussed. Display managers 46 display various process graphics and other information on monitor 26. The KAGUI may call these various process graphics and system information to be displayed on the monitor 26. Pointing device 24, such as a mouse, is used to select keys in the KAGUI or other graphical elements of the KAGUI, and to otherwise provide input to the process control system.

Alarm alert task 30 maintains a state table of all alarm information generated by the process control system. Alarm server task 42 processes commands to be executed by the process control system. Some of these commands originate with the KAGUI, others originate from process control interactive displays that appear on display monitor 26. Alarm alert task 30 communicates over IPC path 32 to inform server task 34 whenever a change occurs in the alarm state of the process control system (e.g., a sensor indicates an alarm, or a sensor in alarm returns to normal). The server then communicates this change to the client task 38, and the presentation of alarm data by the KAGUI is updated.

Occurrence of an alarm is generally accompanied by the sounding of a horn, or the generation of whatever audible stimulus the user has selected for that alarm. Hardware annunciator keyboard 50 is included in FIGURE 2 to illustrate that, if desired, it can be used in addition to the KAGUI.

A graphical user interface (GUI) is a powerful, interactive display format that enables a user to operate a computer by pointing to and selecting pictorial representations on a screen. The modern GUI is a considerable improvement over the text-based entry operating systems of just a few years ago, which required a user to remember several cryptic commands just to make a computer perform a simple task. A typical process control system has its own GUI that displays, for example, process graphics. The KAGUI of the present invention can be implemented by providing keypad annunciator functionality through the GUI interface of the process control system.

FIGURE 3 illustrates a typical process graphic 56, displayed on monitor 26, of a process for manufacturing coke. FIGURE 4 illustrates the KAGUI implementation of a traditional hardware annunciator keyboard (HAK) 60 displayed with a processing control system process graphic 58. Process graphic 58 appears in the upper portion of FIGURE 4. The KAGUI implementation of the hardware annunciator keyboard 60 appears in the lower portion of FIGURE 4. The KAGUI implementation of the hardware annunciator keyboard 60, referred to herein as a "soft HAK", is composed of a numeric entry keypad 64 and two alarm button panels 66 and 68, respectively.

Also included is button 70, for silencing an alarm horn. Panels 66 and 68 are composed of individual keys. There are sixteen keys per panel arranged in four columns and four rows of keys per panel. In the soft HAK illustrated in FIGURE 4, not all the keys are associated with an alarm state. Each key associated with an alarm state, however, has a label displayed on the key, such as on key 72. A pointing device, such as a mouse, is used by the operator to select keys. Operator input via a light pen, or a touch screen, is also well-known in the art.

The KAGUI of the present invention allows an operator to readily configure the layout of a keypad annunciator. The operator may customize the keypad annunciator to better present alarm data or to better allow control of the process control system. FIGURE 5 illustrates two examples of customized keypad annunciators. In FIGURE 5A, keypad annunciator 72 has three panels of keys (panels 74, 76 and 78). The operator has chosen to eliminate the numerical keypad 70 of the soft HAK illustrated in FIGURE 4. The use of three panels of alarm keys accommodates additional alarm states. FIGURE 5B illustrates another customized keypad annunciator. In keypad annunciator 80, there is only one panel of eight keys. The silence horn button 70, included in keypad annunciators 72 and 60, is not included in keypad annunciator 80. The KAGUI does not limit the operator to one keypad annunciator only; the operator can configure the KAGUI to support multiple keypad annunciators. Keypad annunciator 72 of FIGURE 5A and keypad annunciator 80 of FIGURE 5B are both readily incorporated by the KAGUI. This is an advantage of the invention over the prior HAK; incorporation of another keyboard would have required another physical bulky HAK.

Other operator-configurable KAGUI keypads include: a keypad annunciator with 48 buttons arranged in twelve columns and four rows; a keypad annunciator including a numeric keypad, and with thirty-two alarm keys arranged in eight columns and four rows; a keypad annunciator with forty-eight keys arranged in six columns and eight rows; and a keypad annunciator with one to three panels of keys and with up to sixteen keys on each panel.

Based on the disclosures herein, one of ordinary skill in the art can design a configurator appropriate for configuring the KAGUI. Such a configurator can be predominantly "hard-coded," such that most display and other options are fixed, or can provide configuration options that an operator can readily select using, for example, dialog boxes.

Typically, the appearance of a key for indicating no alarm, the occurrence of an alarm and an acknowledged alarm can be configured. Typically a key not indicating an alarm appears in a solid color; a key indicating an alarm appears both in a second color and blinking; and a key indicating an alarm that has been acknowledged by the operator appears in the second color and not blinking. The operator may choose the colors he or she

desires for each key and each state. Also configurable are: The alarm state or states of the process control system to associate with each key; an audible stimulus to be generated when a key is in alarm; actions to be initiated by selecting a key, including, for example, running a program, displaying a process graphic or silencing the audible stimulus; and other display options for the KAGUI, including display of panel manager and dynamic icon, both to be subsequently discussed. Advanced configuration options are also provided, such as a drag-and-drop key configuration and auto block extract from existing display files. The operator selects a pre-configured display file using the file manager and drops the file onto the desired keypad annunciator key. The key is assigned to the display file and all the blocks used in the display are automatically included in the block list for that key. Additionally, the configurator exports and imports the comma separated variable (CSV) format, allowing configuration using third-party tools, such as spread sheets or databases.

The KAGUI presents alarm data to an operator and allows an operator to respond to the occurrence of an alarm state by selecting the appropriate keys. However, the operator will not always display a key pad annunciator as a topmost window on the monitor. It is critical, however, that alarm data be presented and responded to in a timely manner. Accordingly, the invention incorporates several features to enhance operator awareness of alarms. One feature is a compact visual representation of the key pad annunciator, or annunciators, for which the KAGUI is configured. The compact visual representation, referred to herein as a panel manager, is illustrated in FIGURE 6.

Panel manager 81 in FIGURE 6 includes graphical elements, or buttons, for each keypad annunciator to which it is responsive. Panel manager thus represents four keypad annunciators and has four buttons: General overview button 82, boiler overview button 84, tank overview button 86 and boiler display button 88. Occurrence of an alarm state associated with a key of a given keypad annunciator causes the appropriate button, responsive to that keypad annunciator, to illuminate. Selecting the illuminated button with the pointing device causes the keypad annunciator to appear as a high priority window on the monitor. The user then deals as necessary with the alarm.

The operator may configure panel manager 81. Typically a solid button color indicates no alarm, a second blinking color indicates an alarm, and a second, solid color indicates an acknowledged alarm. The panel manager 81 is typically larger than an icon but smaller than a keypad annunciator, however its exact size relative to a keypad annunciator depends on the number of keypad annunciators that the panel manager 81 manages. The panel manager is configurable to float on top of the desk top as a high priority window or appear on the display the occurrence of an alarm state associated with keys of selected keypad annunciators. FIG. 7 illus-

trates a process graphic for a centrifuge 92 over which the panel manager 81 floats.

Should the operator desire to fully minimize the KAGUI, the invention provides a dynamic icon 95. The icon can be configured to float on the desk top as a display with a priority higher than most, or all, windows, or to display upon occurrence of an alarm state. The icon 95 indicates alarms by dynamically changing its appearance. Dynamic icon 95 is illustrated in FIGURE 8A and 8B. Upon occurrence of selected alarm state, the icon appears as continually changing from icon state 94 to icon state 96; otherwise, the icon is static, and can appear as in FIGURE 5A or as in FIGURE 5B.

The invention thus provides an operator with a wide range of display options.

FIGURE 9 illustrates dialog 97 for selecting run time display options of the KAGUI. Option boxes 98 and 102 are for selection of display options for the panel manager and keypad annunciators, respectively. In panel manager option box 98, option button 100 and check boxes 104 and 106 are selected. Upon the occurrence of an alarm state, the panel manager, if obscured by other windows, is brought to the front of the display. If the panel manager 81 is minimized, the dynamic icon is brought to the front of the display. In option box 102, option button 110 is selected, therefore, the keyboard annunciators do not float on the display as high priority windows and do not appear upon occurrence of an alarm state.

Note that the operator may choose to eliminate all visual cues to the occurrence of an alarm by choosing option button 108 in panel manager option box 98 and option button 110 in option box 102. In this case an operator is relying on the horn or other configured audible stimulus to warn him of an occurrence of an alarm. Note that selecting option button 112, "float on top," in panel manager box 98 and option box 110, "do nothing," in option box 102, float the panel manager on the top of the display, and minimizing the panel manager 81 brings the icon 94 up as a floating display.

FIGURE 10 illustrates a flow chart for the runtime behavior of the KAGUI in alerting the operator to the occurrence of an alarm state, based on options selected in dialog 97. Normally the KAGUI continually tests for the occurrence of alarm states, as in 116. If no alarm states are occurring, the KAGUI periodically tests the communication path, as indicated by box 118. Box 118 will be elaborated on subsequently. Upon occurrence of an alarm state, the KAGUI generates the audible stimulus selected for that alarm state, as indicated in box 120. Although not recommended, an operator can eliminate all visual and audible stimuli. Typically the operator will be alerted to the occurrence of an alarm state by both audible and visual stimuli.

From box 120 the KAGUI proceeds to decision box 122, and determines whether the keypad annunciator containing the key in an alarm state is displayed. If the keypad annunciator is displayed, the key is illuminated in the selected manner. If the keypad annunciator is not

displayed, the KAGUI proceeds to decision box 126. If the keypad annunciator is configured to come to the front of the display as a high priority window on alarm, the keyboard annunciator is displayed, as indicated by box 128, and the key is displayed in the selected manner as in box 124. If the key pad annunciator is not configured to appear on alarm, the KAGUI proceeds from decision box 126 to box 129, and as indicated by that box, option box 110 in key pad annunciator option box 102 of FIG. 9 has selected, and the KAGUI "does nothing" with the keypad annunciator displays. Both box 129, "do nothing" and box 124 "display key pad annunciator in selected manner" bring the KAGUI to decision box 130.

If the panel manager 81 is selected as floating, the KAGUI proceeds along the "yes" branch to decision box 132. From box 132, if the panel manager is minimized, the KAGUI proceeds from box 132 to box 134 and the icon dynamic behavior is shown. If the panel manager 81 is not minimized the KAGUI proceeds from box 132 to box 136 and the appropriate button, for example, button 84 of panel manager 81 is displayed in the selected manner. Both box 134, "show dynamic icon behavior," and box 136, "display the panel manager button in selected manner," lead the KAGUI to box 150, where the KAGUI awaits operator input.

Returning to decision box 130, if panel manager 81 is not selected as floating, the KAGUI proceeds instead along the "no" branch to decision box 138. If the panel manager 81 is configured in dialog 97 to appear, the KAGUI proceeds along the "yes" path from box 138 to decision box 140. From 140, if the panel manager is not minimized, the KAGUI proceeds to box 142, displays the panel manager and then proceeds to box 136 and displays the appropriate button of panel manager 81 and then proceeds to box 150, "respond to operator input." If box 140 decides that the panel manager is minimized, the KAGUI proceeds to decision box 144, and determines whether the panel manager is selected to appear when minimized, that is, it determines whether option button 100 on property page 97 is selected and whether check box 106 is checked. If the answer is yes to both, the KAGUI proceeds to box 146, "display icon," then to box 134, "show icon dynamic behavior," and then to 150, "respond to operator input." Otherwise, following the "no" branch decision box 138, the KAGUI proceeds to box 148, and then to box 150, "respond to operator input." Note that the operator may configure the display, such that no visual stimuli are produced on an occurrence of an alarm.

Described in FIGURE 10 is a particular flow chart for a KAGUI to follow in producing visual stimuli. It will be apparent to those of ordinary skill in the art that sequences of steps and decisions in FIGURE 10 are only examples; there can be variations to those steps and sequences illustrated in FIGURE 10, including, for example, following more or less than all the steps of FIGURE 10, modifying one or more of the steps, or changing the order of some or all steps, without departing from

the spirit or scope of the invention. These variations are therefore considered a part of the present invention.

The KAGUI alerts the operator to the occurrence of selected alarm states of the process control system and enhances the operator's ability to respond. However, for the KAGUI to be effective it must accurately represent the alarm state of the process control system. Individual computers or processors, and processes running on computers, communicate by way of IPC paths. Referring to FIGURE 2, the alarm alert task 30 communicates with server task 34 over IPC path 32, and server task 34 communicates with the client task 38 over IPC path 36. The alarm alert task 30 maintains a state table of all alarm data generated by the process control system, and typically is a computational process running work station 12. If any entry in the state table changes, for example, because of an alarm being generated, the alarm alert task communicates this change in alarm state over IPC path 32 to server task 34. Server task 34 in turn routes this change in alarm state to client task 38 by communicating over IPC path 36. The alarm state is then displayable by the KAGUI.

The goal of the process operator is a healthy running industrial process in which alarm states are not often generated. In practice, IPC paths 32 and 36 may not be used regularly, and therefore are not frequently tested. However, it is imperative that when a change in alarm state does occur, the IPC paths 32 and 36 are capable of transmitting this change in alarm state.

Several types of IPC paths, including pipes and sockets, are well known to those skilled in the art. In addition, a given process control system may use a proprietary type of IPC path. IPC paths can further be categorized as connectionless and connection-oriented. In connectionless IPC path, the sending of data in one direction on a path, for example, from alarm alert task 30 to server task 34 over IPC path 32, causes an acknowledgment to be sent from server task 34 back to alarm alert task 30. In response to that first acknowledgment a second acknowledgment is sent from alarm alert task 30 to back to server task 34. The acknowledgment, however, does not usually contain information that identifies the data packet being acknowledged. A connection oriented IPC path is a little more sophisticated. If, for example, server task 34 communicates with client task 38 over a connection oriented IPC path 36, each data packet traveling from server task 34 to client task 38 generates an acknowledgment packet from client task 38 to server task 34. The acknowledgment packet contains a data sequence that identifies which data packet is being acknowledged. If server task were to send data and no acknowledgment is received by the server task within a specified time, the server task automatically resends the data packet over the IPC path 36.

Connectionless IPC paths are often suitable when two processes that are communicating are both running on the same machine or computer. The more sophisticated connection oriented path is used when two proc-

esses running on different machines communicate over a network connection, as network connections may not be as reliable as interprocess communication on the same processor.

The preferred embodiment of the invention employs connection oriented IPC paths and include further features to maximize the probability that the alarm state information displayable by the KAGUI is accurate and to inform an operator of the status of IPC paths 32 and 36 used to communicate alarm state information.

According to this feature, the invention maintains IPC paths used for the transmission of alarm data by continually sending verification messages over the IPC paths. The content of these verification signals is typically irrelevant; their purpose is to generate acknowledgment signals from the receiver of the verification message. The entities at each end of a given IPC path both transmit verification signals to verify the continued existence of the path, and, should the path fail to function, i.e., should a failure to receive acknowledgment signals be detected, they cooperate in reestablishing an IPC path. For example, server task 34 in FIGURE 2 constantly sends verification signals over IPC path 36 to receive a constant stream of acknowledgment signals from client task 38. Client task 38 sends a constant stream of verification signals over path 36 to receive a steady stream of acknowledgment signals from server task 34. If the path 36 fails, the server task 34, upon detecting a failure to receive acknowledgment signal, usually by waiting for a selected period of time after sending verification messages, goes into listening mode. The client task 38, similarly detecting a failure of the communication path 36, goes into an attach mode. Together, the server 34 and client 38 re-establish a new communication path 36.

The operator is alerted to the status of communication path 36 by a graphical element that is part of panel manager 81. Referring to FIGURE 6, graphical element 90, in the illustrated embodiment a tool button, simulates an LED. Toolbutton 90 blinks green as long as all IPC path being verified are open and functioning, that is, as long as regular acknowledgment signals are received. Should a failure of an IPC path be detected, the tool button 90 blinks red to alert an operator that the alarm state information presented by the KAGUI might not be accurate.

The invention also includes an automatic synchronization procedure for maximizing the probability that the KAGUI displays the proper alarm state condition. This synchronization technique is discussed in conjunction with FIGURE 11, which also illustrates the run time behavior of the KAGUI. Referring to FIGURE 11, the KAGUI continually checks for the occurrence of alarm states and awaits operator input, and indicated by boxes 115 and 116. Should an alarm state be generated, the KAGUI proceeds to box 120 and generates the selected audible stimulus. Proceeding to decision box 152, if the KAGUI is configured for visual stimulus, the KAGUI dis-

plays the selected visual stimulus as indicated by box 154. The KAGUI then responds to operator input as indicated by box 156. If the KAGUI is not configured for the display of visual stimuli, it proceeds directly from box 152 to box 156. In either case, the KAGUI responds to operator input until the problem is fixed, as illustrated by decision box 158.

If the test for the occurrence of an alarm state in decision box 116 indicates that no alarms states have changed, the KAGUI automatically proceeds to box 118 and tests communication paths. Verification signals are continually sent over important IPC paths. Should all acknowledgments be received, as indicated by the yes branch of decision box 160, the KAGUI simply returns to its normal state, as in decision boxes 115 and 116, of responding to operator input and continually testing for the occurrence of an alarm state. Should a failure to receive acknowledgment messages be detected by any sender, the KAGUI proceeds to box 162 and alerts the operator, typically by changing toolbutton 96 from green to red. After alerting the operator, the KAGUI reestablishes the pipe or socket or other proprietary communication path between, for example, server task 34 and client task 38, as indicated by box 164. Once the IPC path is re-established, the KAGUI resynchronizes the alarm data presented by the KAGUI with current alarm data by downloading, as indicated in box 166, all alarm state data from alarm task 30 to client task 38, which is associated with the KAGUI. The KAGUI alarm data is thus refreshed. Should server task 34 and client task initially fail to establish a communication path, they continue to try until a path is established. The KAGUI then proceeds to box 168, and the operator is informed, by panel manager toolbutton changing back to a simulated green LED, that IPC communication paths are now operational. The KAGUI then returns to its normal state of responding to operator input and continually testing for an alarm state, as indicated by boxes 115 and 116. Note that synchronization need not be automatic; the operator could be relied upon to manually download data.

It is expected that one of ordinary skill in the relevant arts, possessed of the teachings herein, can create a KAGUI for use with a process control system using an object oriented approach in a computer language such as C++. Implementation of the KAGUI described herein relied heavily on published class libraries such as the Microsoft Foundation Class library. However, the visual basic computer language should also allow design of a key pad annunciator graphical user interface quite similar to that disclosed herein.

The invention thus provides a keypad annunciator graphical user interface (KAGUI) for presenting an interactive keypad annunciator on a monitor display. Alarm state data are presented, and an operator can respond by selecting elements of the keypad display, such as a key, with a pointing and selecting input device. Keypad annunciators can be readily configured to allow efficient and effective management of the process control

system. The KAGUI includes a panel manager and a dynamic icon to enhance operator awareness, in a multi-window display environment, of alarms. Many display options for keypad annunciators, the panel manager, and the icon are available. The KAGUI synchronizes alarm data presented by the KAGUI with process control system alarm data and informs the operator of the status of interprocess communication paths.

It will thus be seen that the invention efficiently obtains objects set forth above among those made apparent from the preceding description. Because certain changes may be made in the above constructions without departing from the scope of the invention, it is intended that all matter contained in the above description and shown in the accompanied drawings be interpreted as illustrative and not an limiting sense.

It is also to be understood that the following claims are intended to cover generic and specific features of the invention described herein and all statements of the scope of the invention which as a matter of language might be said to fall therebetween.

Claims

1. A process control system, including at least one sensor for determining the state of at least one process variable, a processor means for generating alarm state data when the variable is in a predetermined state, and a display monitor,

comprising display means for displaying on the display monitor at least one keypad annunciator having at least one key displayable in at least two key display states, said display means displaying said one key normally in a first key display state and being responsive to alarm state data for displaying one key in a second of the at least two key display states upon occurrence of a selected alarm state, and input means operative with said display means for changing a display, on the monitor, associated with the keypad annunciator, and for responding to the occurrence of the alarm state.

2. A process control system according to Claim 1, further comprising

panel manager means for displaying, in an area normally occupying a portion of the display screen of the display monitor, a panel manager for visually representing a selected condition of alarm state data associated with a keypad annunciator, and

input means operating with said panel manager means for changing a display, on the display monitor, associated with said panel manager.

3. A process control system according to Claim 2 further wherein

said panel manager includes floating panel manager means responsible to a user input for displaying said panel manager on the monitor display such that the display of said panel manager has a priority higher than a majority of other windows.

4. A process control system according to Claim 2 further comprising

dynamic icon means for displaying on said display monitor a dynamic icon, having at least two dynamic icon display states, said dynamic icon display means displaying said dynamic icon normally in a first dynamic icon display state and being responsible to selected alarm states of the process control system, for displaying said dynamic icon in a second dynamic icon display state, said dynamic icon means responsive to the selection of said dynamic icon by way of said input means for displaying the panel manager on the display in response to a selected input.

5. A process control system according to Claim 1 further comprising

panel manager means for displaying a panel manager in an area normally occupying a portion of the display screen of the display monitor, said panel manager means being responsive to at least one selected alarm state and being associated with at least one keypad annunciator, said panel manager having at least two unique display states for each keypad annunciator with which said panel manager is associated, said panel manager, responsible to the occurrence of an alarm state that is associated with said at least one keypad annunciator with which said panel is associated, displaying said panel manager in the second panel manager display state associated with that keypad annunciator, and

dynamic icon means for displaying on said display monitor a dynamic icon, having at least two dynamic icon display states, said dynamic icon display means displaying said dynamic icon normally in a first dynamic icon display state and being responsive to selected alarm states of the process control system, for displaying said dynamic icon in a second dynamic icon display state, said dynamic icon means responsive to the selection of said dynamic icon by way of said input means for displaying the panel manager on the display in response to a selected input.

6. A method of reporting the occurrence of an alarm state within a process control system, comprising

the steps of

displaying at least one keypad annunciator on a display monitor,
displaying at least one key of the keypad annunciator normally in a first key display state, displaying the at least one key in a second display state upon occurrence of a selected alarm state, and
changing a display, on the display monitor, associated with the keypad annunciator, and responding to the occurrence of the alarm state in response to an input device.

7. A reporting method according to Claim 6, further including

displaying a panel manager in an area normally occupying a portion of the display screen of the display monitor,
associating the panel manager with at least one keypad annunciator,
providing the panel manager with at least two unique display states for each keypad annunciator with which the panel manager is associated, and
displaying the panel manager in a second panel manager display state associated with a keypad annunciator upon occurrence of an alarm state associated with a key of that keypad annunciator.

8. A reporting method according to Claim 7, further including

displaying said panel manager on the monitor display as a display having a priority higher than a majority of other windows upon the occurrence of at least one selected alarm state.

9. A reporting method according to Claim 7, further including

displaying on said display monitor a dynamic icon,
displaying said dynamic icon normally in a first dynamic icon display state,
displaying said dynamic icon in a second dynamic icon display state in response to the occurrence to selected alarm states, and
displaying the panel manager on the display in response to a selected input by way of selecting the dynamic icon with the input device.

10. A reporting method according to Claim 9, further including

displaying said dynamic icon as a display having a priority higher than a majority of other windows on the display monitor in response to the occur-

rence of at least one selected alarm state.

11. Apparatus for maintaining synchronization between first data representative of a process control system and second presentation data response to the first data, comprising

means for transmitting a succession of verification signals between a computational task of the process control system and a keypad annunciator graphical user interface of the process control system,

means for receiving, under normal conditions, acknowledgement signals in response to successful transmission for verifying the continued existence of a communication path therebetween,

means for maintaining the communication path, including

detecting means for detecting a failure to receive the acknowledgement signals,

establishment means for responding to a detected failure to receive the acknowledgement signals by re-establishing a communication path between the computational task and the keypad annunciator graphical user interface, and

refresh means for transmitting current alarm values of the first data from the computational task to the keypad annunciator graphical user interface to refresh the second presentation data, said refresh means transmitting the current alarm values using the re-established communication path.

12. A method for maintaining synchronization between first alarm data of a process control system and the presentation thereof by a keypad annunciator graphical user interface, comprising the steps of

transmitting a succession of verification signals on a communication path used for the transmission of process control system alarm data, and receiving, under normal conditions, acknowledgement signals in response to successful transmission of the verification signals for verifying the continued existence of the communication path,

maintaining said communication path, including detecting a failure to receive said acknowledgement signals, and in response to a failure to receive acknowledgement signals, re-establishing a communication path, and

transmitting current first alarm data of the process control system to the keypad annunciator graphical user interface, using the re-established communication path, in response to the re-establishment of the communication path.

13. A method according to Claim 12 further including the steps of providing, at the keypad annunciator graphical user interface, a first indication in response to verifying the existence of the communication for transmission of alarm data, and including the step of changing said first indication in response to a failure to verify the existence of the communication path for transmission of alarm data.

14. A method according to Claim 12 further comprising the steps of

(a) refraining, under selected first conditions, from transmitting alarm values of said first data from the computational task to the keypad annunciator graphical user interface

(b) upon a change in an alarm value of the first data, transmitting the changed alarm value from the computational task to the keypad annunciator graphical user interface to update the second presentation data, and

wherein said first conditions exclude the conditions of step (b) and exclude the condition of transmitting current first alarm data of the process control system to the keypad annunciator graphical user interface, using the re-established communication path, in response to the re-establishment of the communication path.

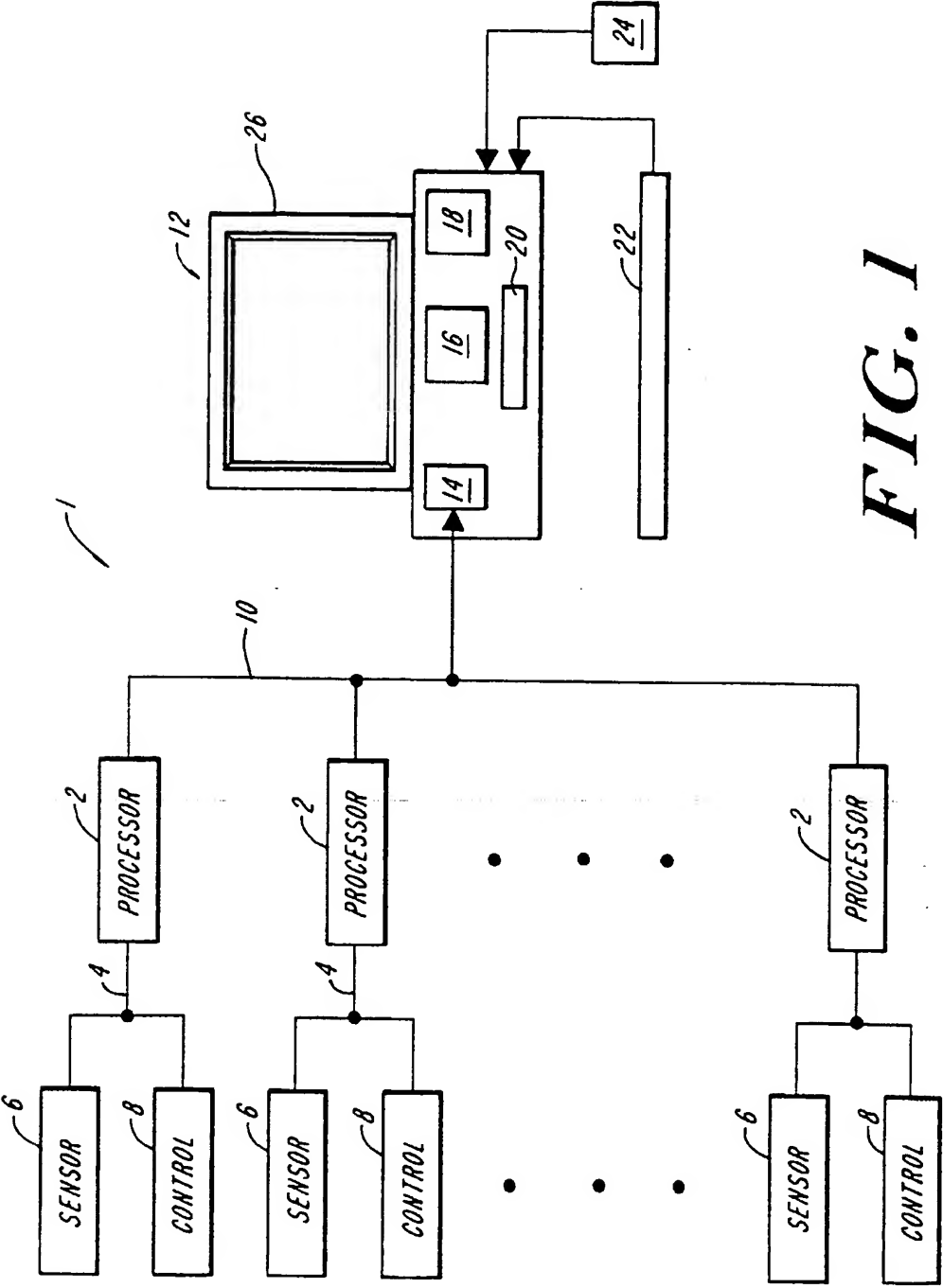


FIG. 1

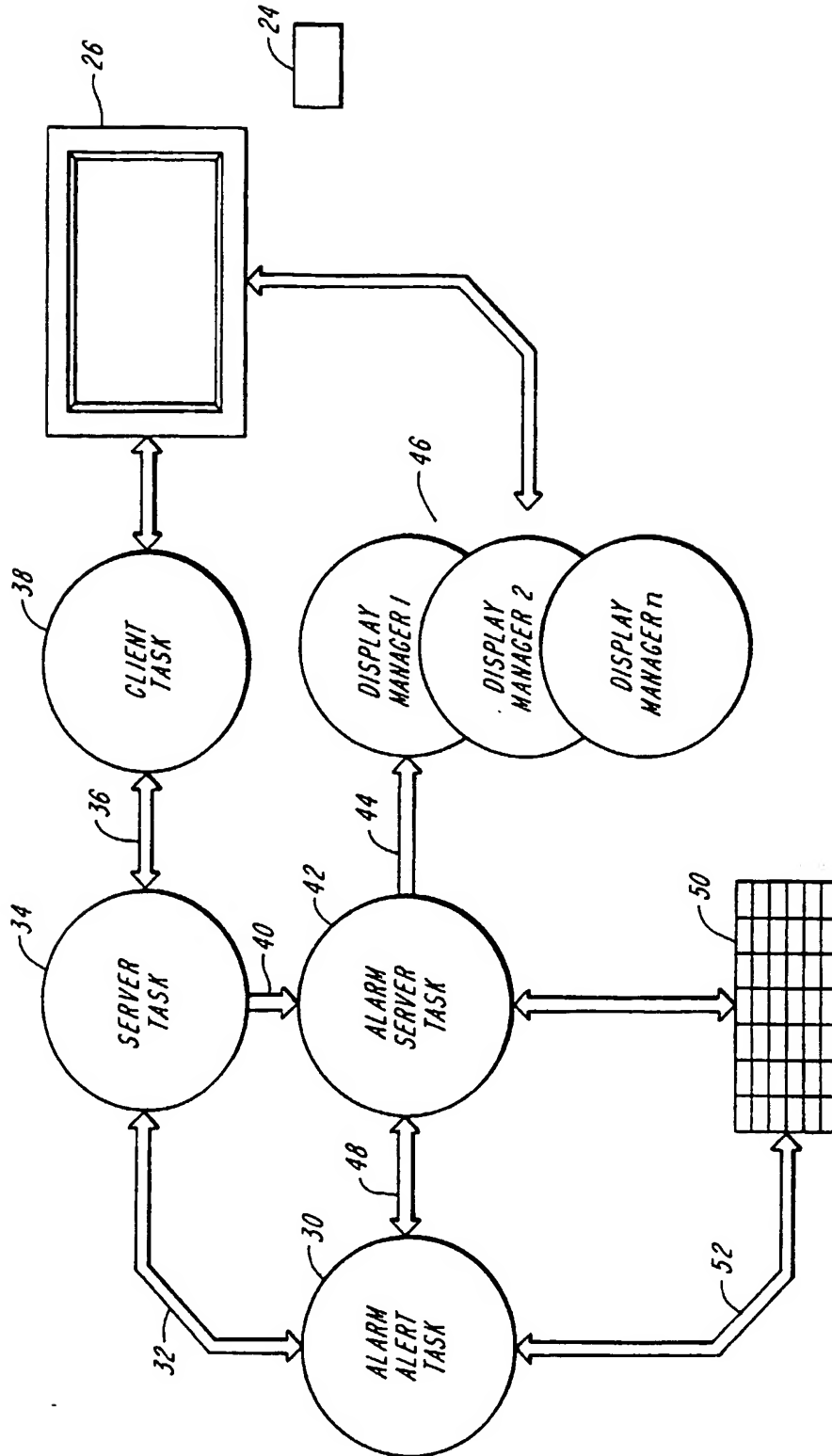


FIG. 2

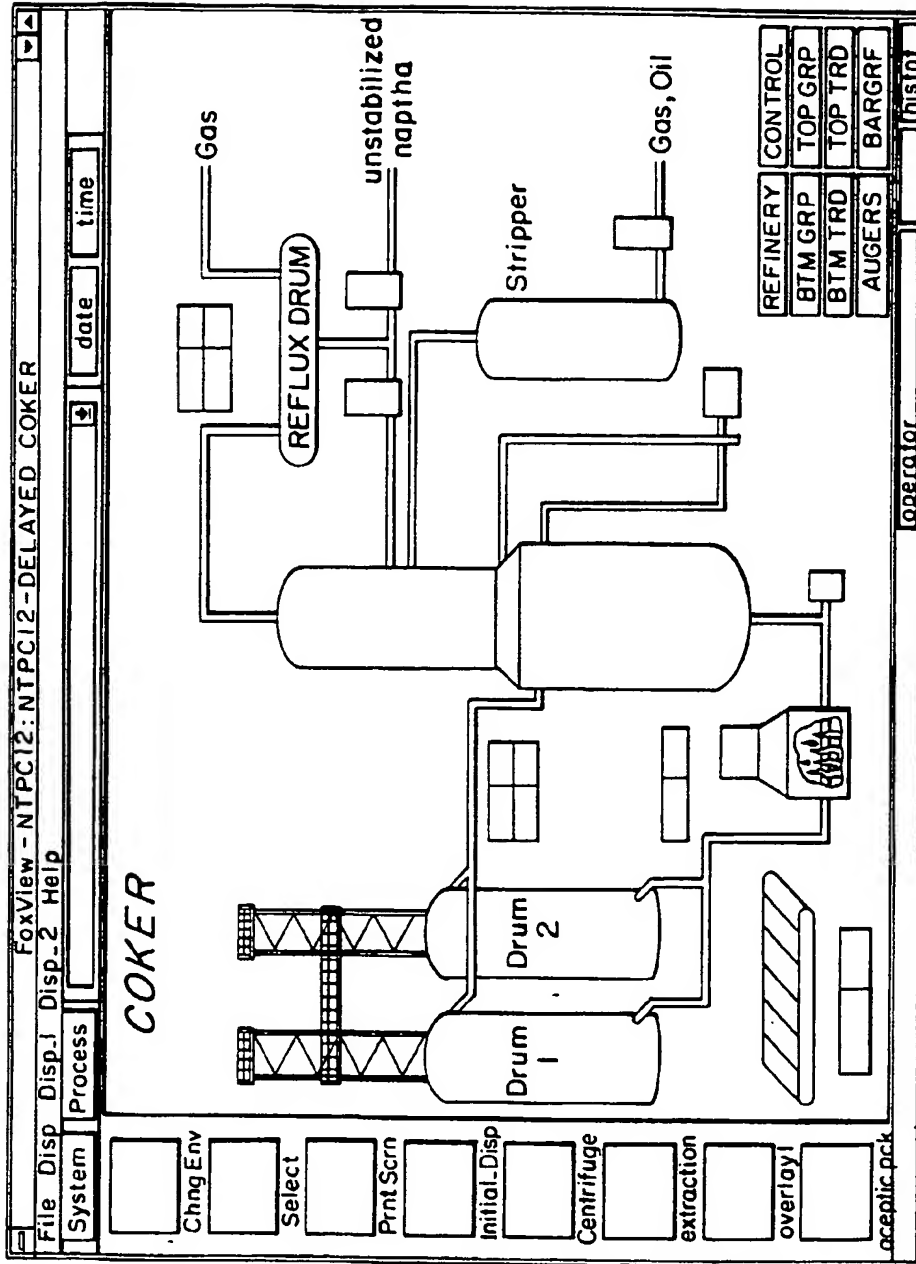
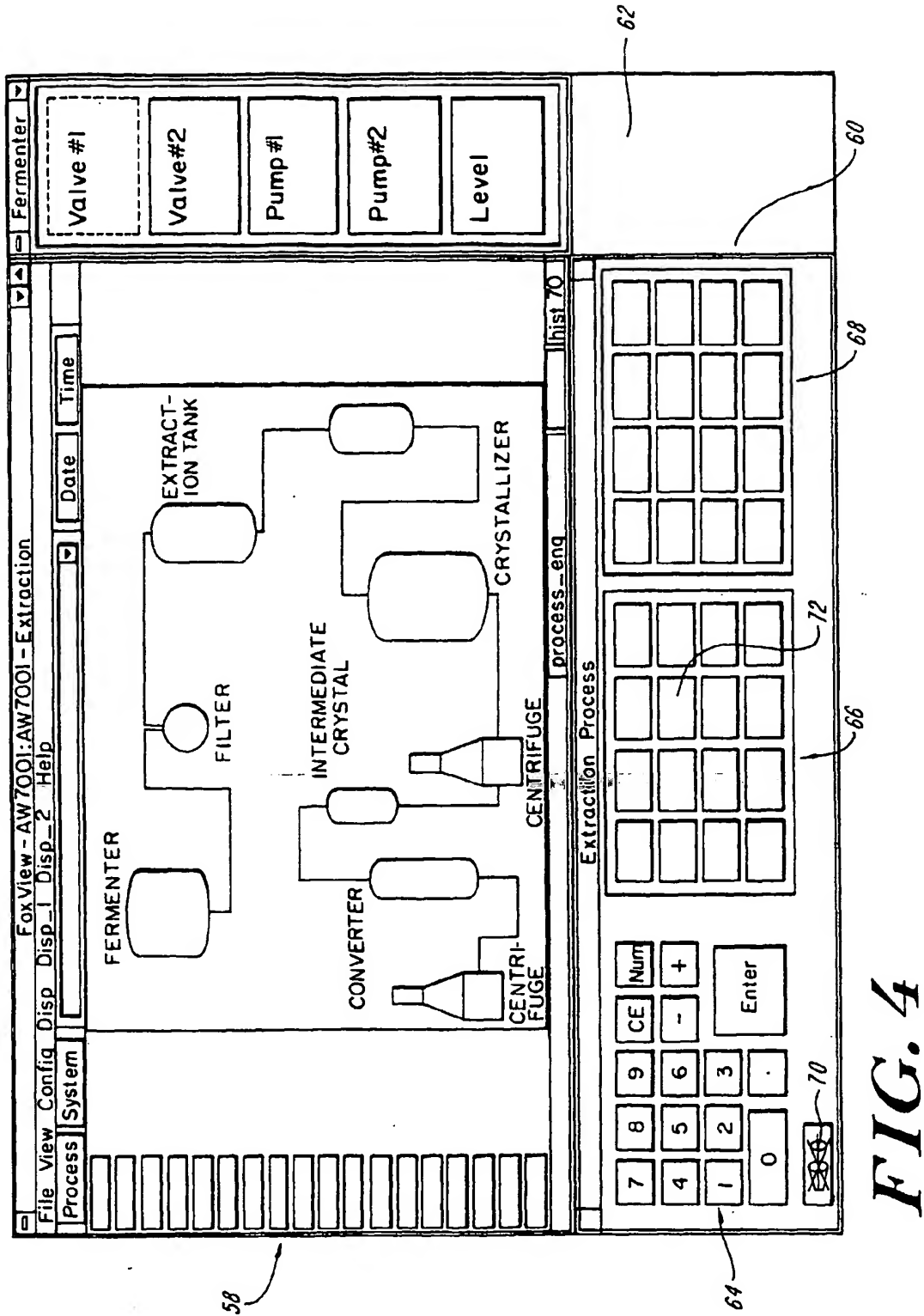


FIG. 3



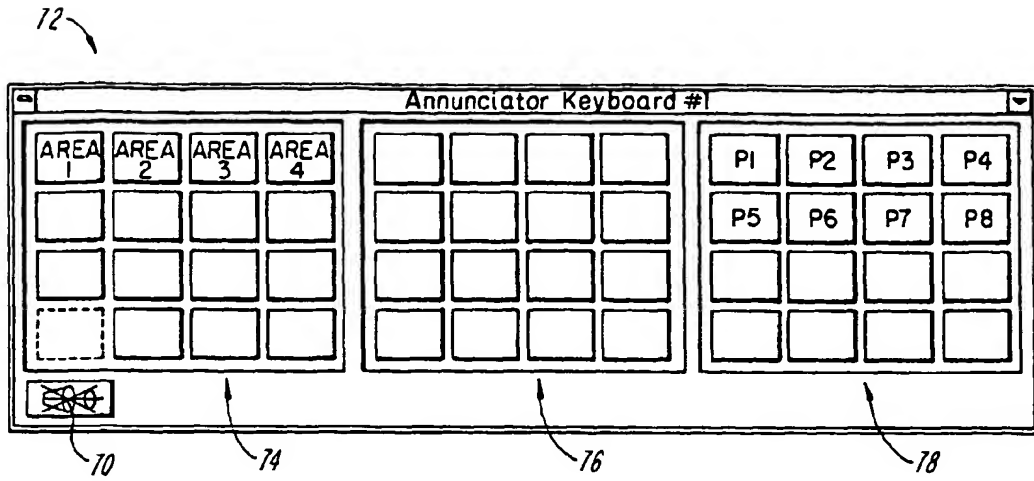


FIG. 5A

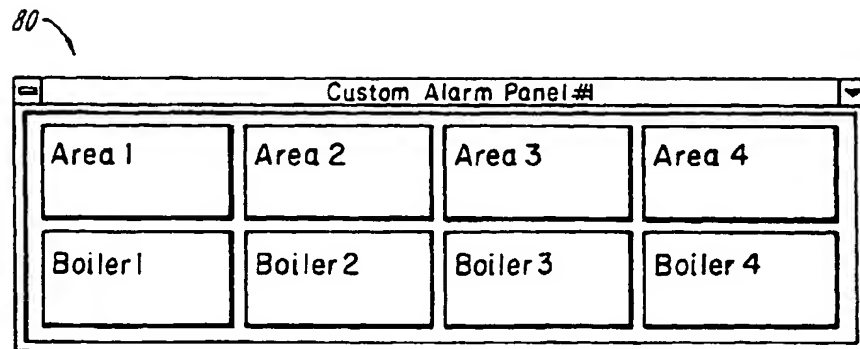


FIG. 5B

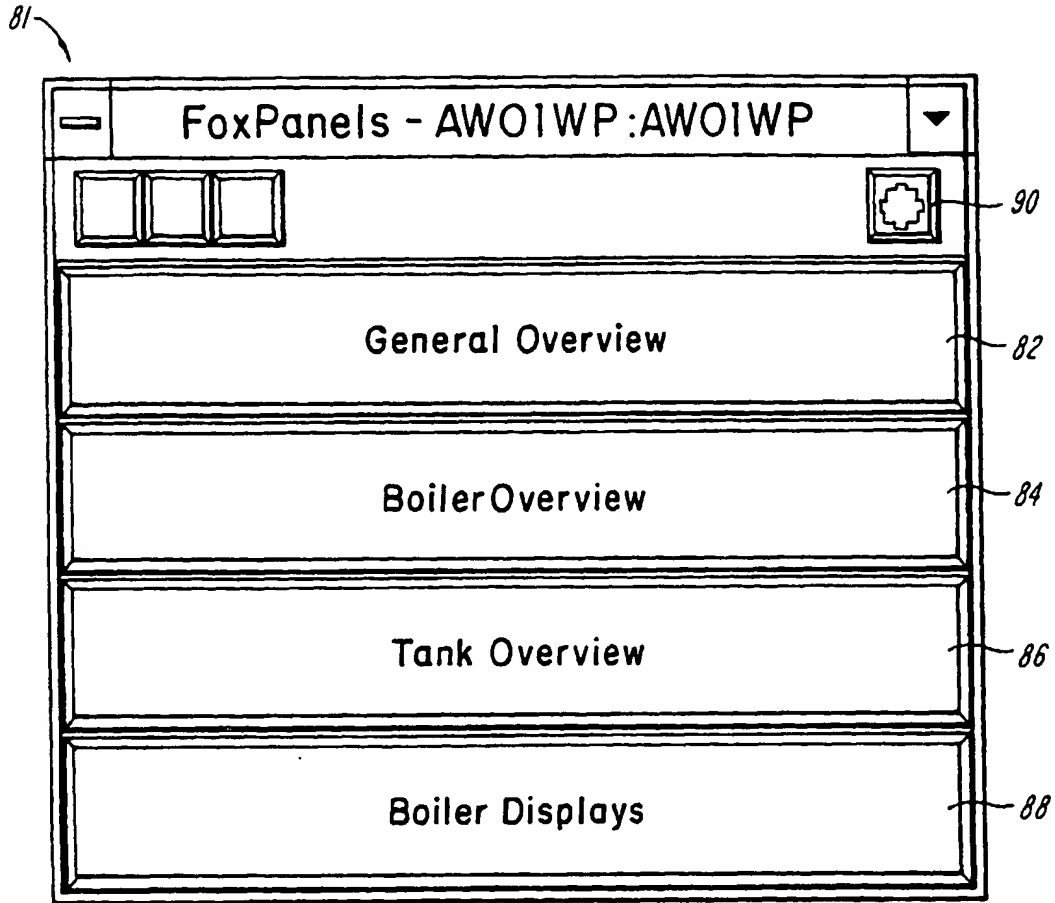


FIG. 6

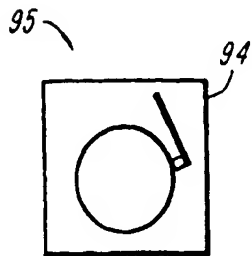


FIG. 8A

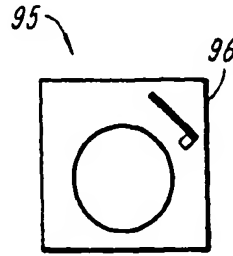


FIG. 8B

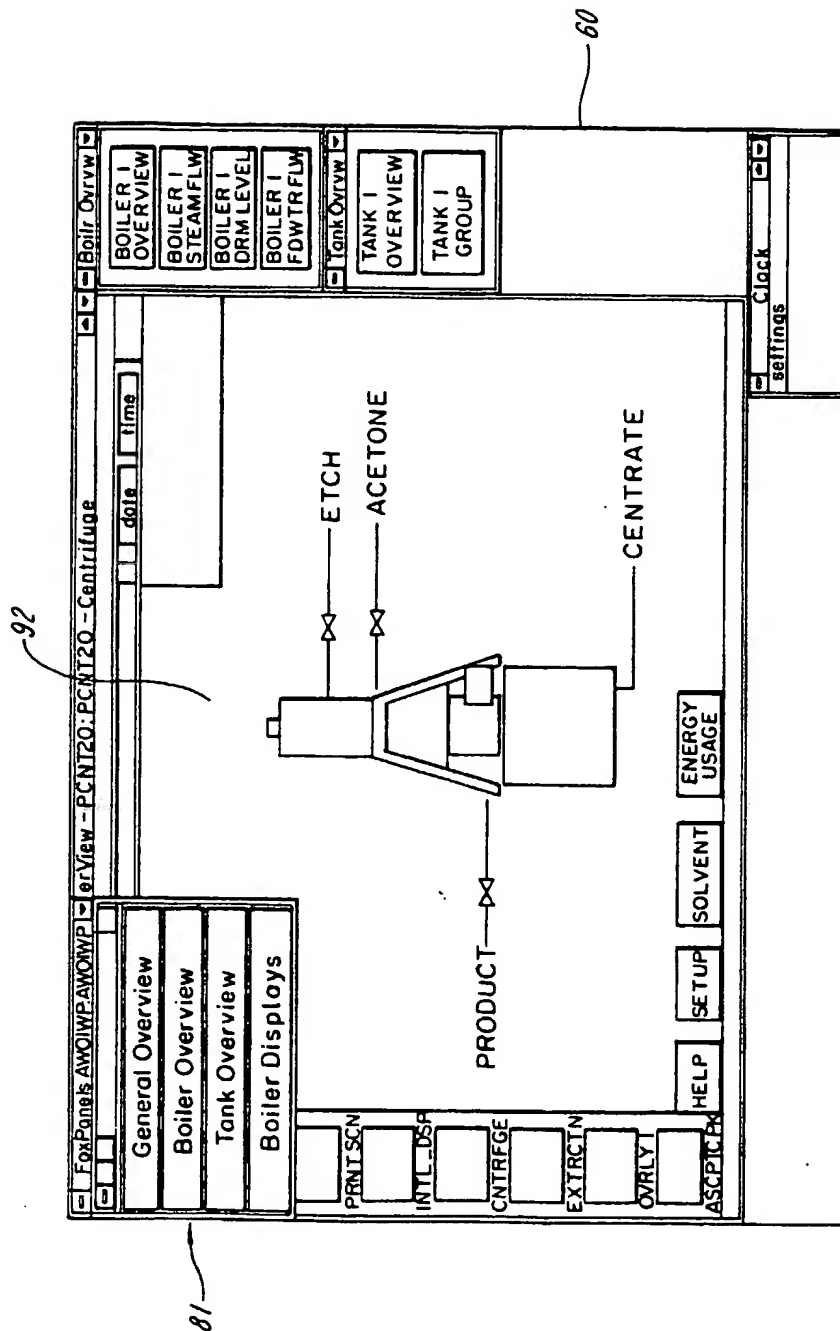


FIG. 7

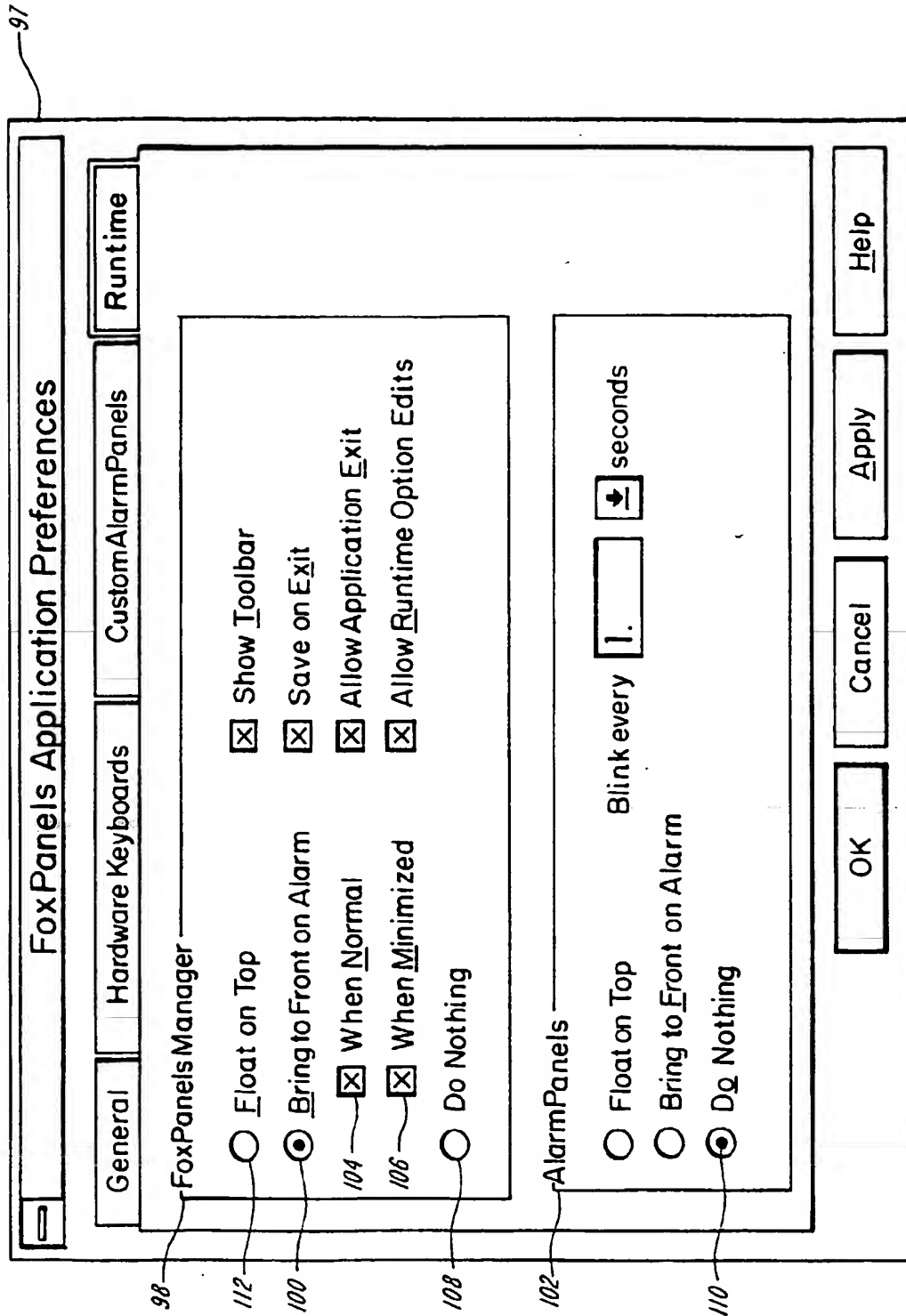


FIG. 9

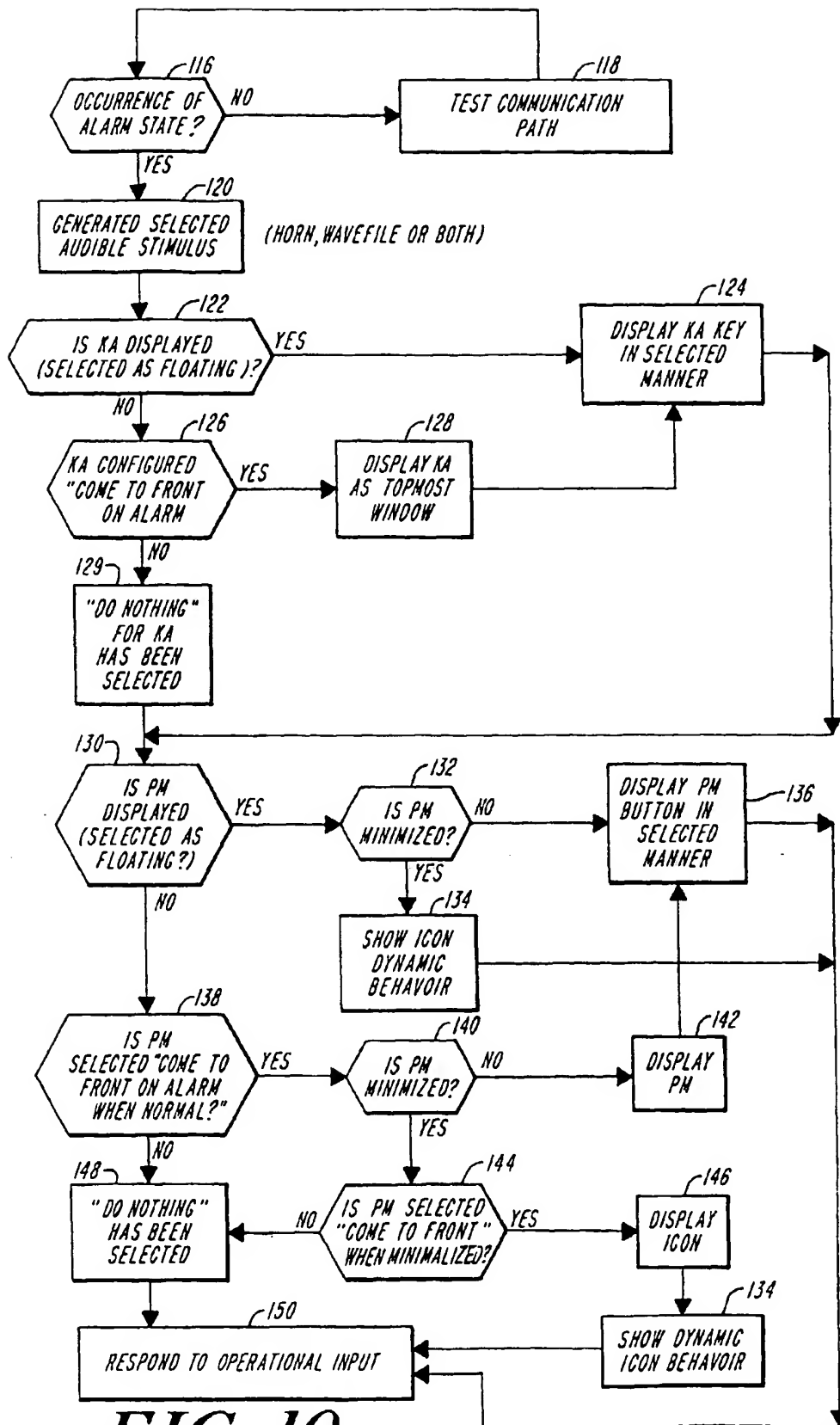


FIG. 10

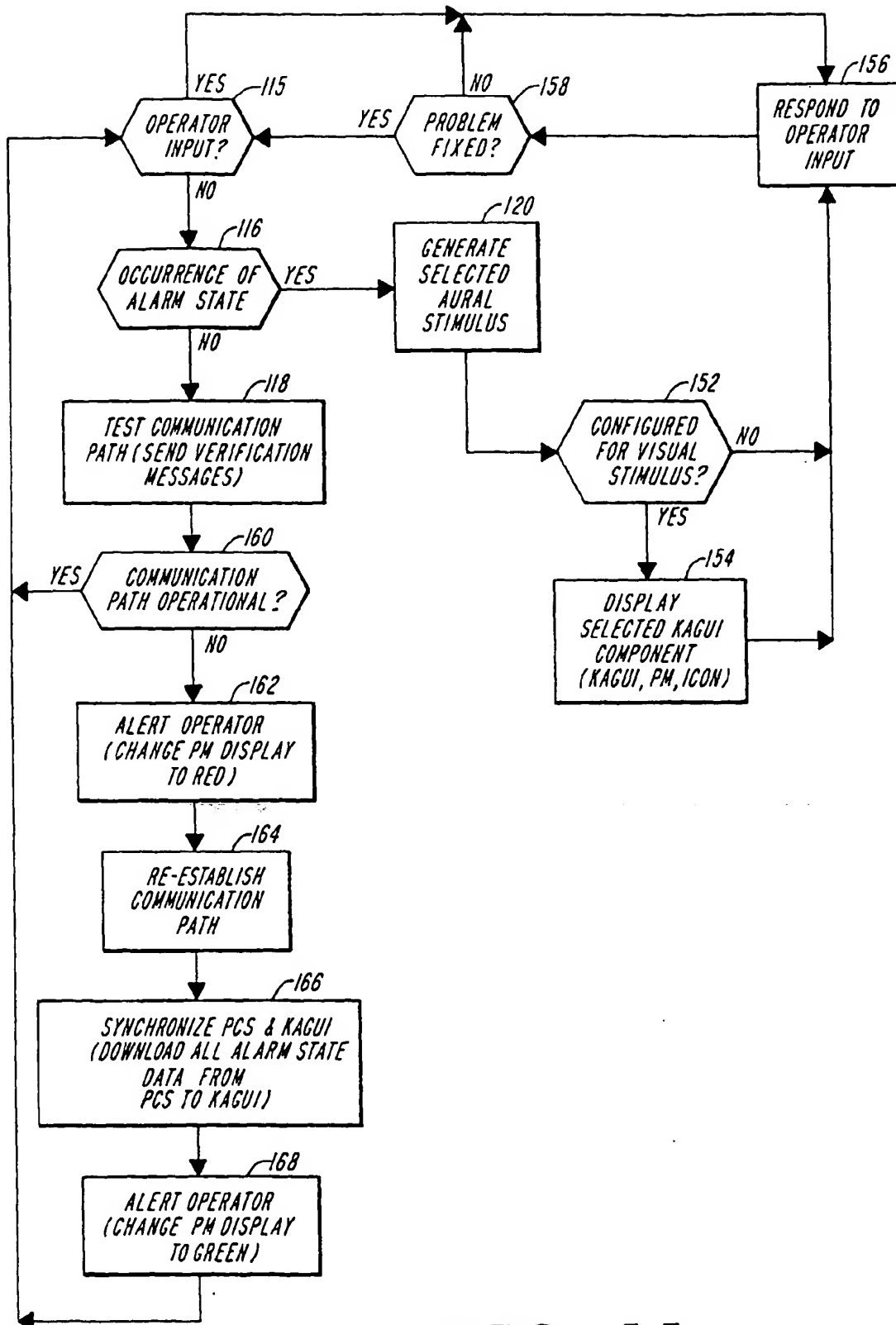


FIG. 11

(19)



Europäisches Patentamt
European Patent Office
Office européen des brevets



(11)

EP 1 122 652 A1

(12)

EUROPEAN PATENT APPLICATION

(43) Date of publication:
08.08.2001 Bulletin 2001/32

(51) Int Cl.7: **G06F 17/30**

(21) Application number: 01102032.8

(22) Date of filing: 30.01.2001

(84) Designated Contracting States:
**AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU
MC NL PT SE TR**
Designated Extension States:
AL LT LV MK RO SI

- Young, Michael J.
Boxborough, Massachusetts 01719 (US)
- DiCelle, Joseph J
Boylston, Massachusetts 01505 (US)
- Wong, David W. H.
Boxborough, Massachusetts 01719 (US)
- Esenther, Alan W.
Ashland, Massachusetts 01721 (US)

(30) Priority: 03.02.2000 US 497610

(71) Applicant: **MITSUBISHI DENKI KABUSHIKI
KAISHA**
Tokyo 100-8310 (JP)

(74) Representative: **Pfenning, Meinig & Partner GbR**
Mozartstrasse 17
80336 München (DE)

(72) Inventors:
• Walsh, Thomas C
Cambridge, Massachusetts 02139 (US)

(54) Data Integration system

(57) An enterprise integration system is coupled to a number of legacy data sources. The data sources each use different data formats and different access methods. The integration system includes a back-end interface configured to convert input data source information to input XML documents and to convert output XML document to output data source information. A front-end interface converts the output XML documents

to output HTML forms and the input HTML forms to the XML documents. A middle tier includes a rules engine and a rules database. Design tools are used to define the conversion and the XML documents. A network couples the back-end interface, the front-end interface, the middle tier, the design tools, and the data sources. Mobile agents are configured to communicate the XML documents over the network and to process the XML documents according to the rules.

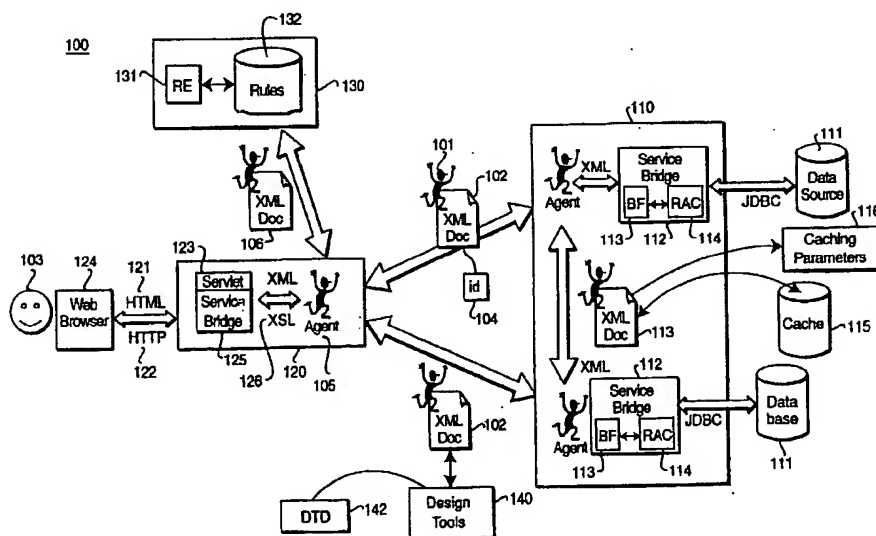


FIG. 1b

EP 1 122 652 A1

Description

Field of the Invention

[0001] This invention relates generally to computerized applications, databases, and interface, and more particularly to integrating applications, databases, and interfaces having different formats, contexts, and designs.

Background of the Invention

[0002] Computer and computer-related technology have enabled the use of computers in numerous enterprise functions. Almost every facet of a modern enterprise is supported by computer systems in some manner. Computerization is a necessity to allow an enterprise to remain functional and competitive in a constantly changing environment.

[0003] Computer systems are used to automate processes, to manage large quantities of information, and to provide fast and flexible communications. Many enterprises, from sole proprietorships, small stores, professional offices and partnerships, to large corporations have computerized their functions to some extent. Computers are pervasive, not only in business environment, but also in non-profit organizations, governments, and educational institutions.

[0004] Computerized enterprise functions can include billing, order-taking, scheduling, inventory control, record keeping, and the like. Such computerization can be accomplished by using computer systems that run software packages. There are many application software packages available to handle a wide range of enterprise functions, including those discussed above.

[0005] One such package is the SAP R/2^(TM) System available from SAP America, Inc., 625 North Governor Printz Blvd., Essington, Pa. 19029. The SAP R/2 System is a software package designed to run on IBM or compatible mainframes in a CICS (Customer Interface Control System) or IMS (Information Management System) environment. For example, SAP may use CICS to interface with user terminals, printers, databases, or external communication facilities such as IBM's Virtual Telecommunications Access Method (VTAM).

[0006] SAP is a modularized, table driven application software package that executes transactions to perform specified enterprise functions. These functions may include order processing, inventory control, and invoice validation; financial accounting, planning, and related managerial control; production planning and control; and project accounting, planning, and control. The modules that perform these functions are all fully integrated with one another.

[0007] Another enterprise area that has been computerized is manufacturing. Numerous manufacturing functions are now controlled by computer systems. Such functions can include real-time process control of

discrete component manufacturing (such as in the automobile industry), and process manufacturing (such as chemical manufacturing through the use of real-time process control systems). Directives communicated from the computer systems to the manufacturing operations are commonly known as work orders. Work orders can include production orders, shipping orders, receiving orders, and the like.

[0008] However, the computerization of different functions within a single enterprise has usually followed separate evolutionary paths. This results in incompatibility between the different systems. For example, transactions from a system for one function may have a context and a format that are totally incompatible with the context and format of another function. Furthermore, as enterprises grow through mergers and acquisitions, the likelihood of inheriting incompatible systems increases. Consequently, the legacy systems cannot provide all the information necessary for effective top level management and control.

[0009] As an additional complexity, enterprise systems need user interfaces for front-end operations. For example, in the healthcare industry, administrative staff and health care providers need reliable access to patient records. If the healthcare enterprise has evolved by a series of mergers, the possibility of a reception desk populated with half a dozen different terminals, each accessing a different patient database and a different accounting system is a certainty, and service and profitability suffers.

[0010] Generic computerized solutions that offer an efficient, automated way to integrate an enterprise's various computerized systems are difficult to implement. Another conventional solution is to implement a custom, computerized interface between the various systems. However, these custom solutions are usually tailored to a specific enterprise environment. As a result, the tailored solutions are not portable into other situations without major modifications. Additionally, these solutions are costly to maintain over time because of inherent difficulties in accommodating change.

[0011] Conventional solutions that meet all of the needs for collecting, retrieving, and reporting data in a complex enterprise do not exist. For example, the DASS^(TM) system, available from a SAP AG, of Waldorf, Germany, is intended to automate manufacturing functions. DASS receives information from SAP R/2 package described above. However, DASS does not appear to provide a generic solution to connect a computerized business system to a computerized manufacturing system.

[0012] Figure 1a shows an example legacy enterprise system 10. The legacy system includes as subsystems a SAP system 11, an Oracle^(TM) database 12, one or more legacy applications 13, Lotus Notes^(TM) 14, a Web server 15, and user interfaces 20. The system 10 might also permit access to some functions by a mobile computer (laptop) 30 via a dial-up communications link 40.

[0013] More than likely, the legacy system 10 will exhibit one or more of the following problems. All sub-systems cannot communicate with every other sub-system because each sub-system has its own application programming interfaces (APIs). Real-time data interchange among all of the sub-systems may be impossible or extremely difficult because each sub-system stores and views data in a different way and uses different communication protocols. Modified enterprise functions or adding automation for new functions is expensive. Each sub-system is developed with its own peculiar programming language. Users cannot always access all the data all of the time, particularly when the user is mobile. It is difficult to provide top level management with an abstraction of all system information.

[0014] What is needed is a system that can integrate various computer systems in an enterprise. The system needs to be able to convey transactional data between any number of databases regardless of their format, context, and access methodology. User interfaces to the databases need to be uniform. In addition, as enterprise functions change, new procedures and transactions must be accommodated in a minimal amount of time without having to redesign and reimplement any of the functional systems. The ideal enterprise integration system should be capable of adapting to any number of computerized functions in a modern complex enterprise.

Summary of the Invention

[0015] The present invention is directed to a system and method for integrating computer systems found in many types of enterprises.

[0016] An enterprise integration system is coupled to a number of legacy data sources. The data sources each use different data formats and different access methods. The integration system includes a back-end interface configured for converting input data source information to input XML documents and for converting output XML documents to output data source information.

[0017] A front-end interface converts the output XML documents to output HTML forms and the input HTML forms to the XML documents. A middle tier includes a rules engine and a rules database. Design tools are used to define the conversion and the XML documents.

[0018] A network couples the back-end interface, the front-end interface, the middle tier, the design tools, and the data sources. Mobile agents are configured to communicate the XML documents over the network and to process the XML documents according to the rules.

Brief Description of the Drawings

[0019]

Figure 1a is a block diagram of a legacy enterprise

system;

Figure 1b is a block diagram of an integrated enterprise system according to the invention;

Figure 2 is a block diagram of design tools used by the system of Figure 1b;

Figure 3 is a block diagram of XML data accesses according to the invention;

Figure 4 is a block diagram of a back-end interface of the system of Figure 1b;

Figure 5 is a diagrammatic of a public interface of the back-end interface of Figure 4;

Figure 6 is a block diagram of pooled connections;

Figure 7 is a flow diagram of a get request;

Figure 8 is a flow diagram of an update request; and

Figure 9 is a block diagram of an object of service bridge objects.

Detailed Description of the Preferred Embodiments

Introduction

[0020] Our invention provides a robust and scalable environment for integrating legacy enterprise computer systems. The invention integrates databases, transactions, and user interfaces having different formats, contexts, and designs, such as the sub-systems shown in Figure 1a. We also provide for automated rules based processing.

[0021] At the core of our integration system, we utilize XML as a universal data encoding and interchange format. XML (Extensible Markup Language) is a flexible way for us to create common information formats and share both the format and the data on the Internet, the World Wide Web (WWW), intranets, and private local area network. XML, developed by the World Wide Web Consortium (W3C), is "extensible" because, unlike HyperText Markup Language (HTML), the markup symbols of XML are unlimited and self-defining. XML is actually a simpler and easier-to-use subset of the Standard Generalized Markup Language (SGML), the standard for how to create a document structure. XML enables us to create customized "tags" that provide functionality not available with HTML. For example, XML supports links that point to multiple documents, as opposed to HTML links, which can reference just one destination each.

These basic interfaces allow our integration system to view, modify and interact with linked legacy applications or legacy data sources.

System Architecture

[0022] As shown in Figure 1b, our enterprise integration system 100 includes the following main components: a back-end interface 110, a front-end interface 120, a middle tier 130, and design tools 140. The components are connected by a network and mobile agents 101 carrying XML documents 102. The mobile agents 101 are described in greater detail in U.S. Patent Application Sn. 08/966, 716, filed by Walsh on November 7, 1997, incorporated herein in its entirety by reference. As a feature, the agents can travel according to itineraries, and agents can "meet" with each other at meeting points to interchange information.

[0023] With our back-end interface 110, we enable read/write/modify access to existing (legacy) applications and data sources 111. The back-end interface maps (or translates) data from legacy formats into the XML format used by our enterprise integration system 100.

[0024] The front-end interface 120 enable us to present information to users 103 using standard presentation methodologies. The front-end interface also allows the user to modify information and to generate transactions to initiate enterprise processes or workflow. The front-end interface can be modified to meet changing requirements of the enterprise.

[0025] The middle tier 130 uses our mobile agents 101 to provide an infrastructure for highly flexible, robust and scaleable distributed applications. The middle tier combines server technology with a customizable business rules engine and an application framework. The middle tier also provides for the deployment of disconnected applications for mobile users. That is, the middle tier allows the mobile user to perform tasks while disconnected from the system 100.

[0026] The design tools 140 support the definition of XML document formats. The design tools also allow us to define mappings of the XML document formats and the legacy data formats, and to provide for the automated generation of forms for user presentation via the front-end interface. These components are now described in greater detail.

Back-End Interface

[0027] The back-end interface 110 is composed of one or more service bridges 112. The service bridges provide highly efficient access to various legacy systems. Hereinafter, we will frequently call the legacy systems "data sources" 111. We do not care how the legacy systems are programmed, or how their applications are structured. That is, the back-end interface of our integration system provides a generic and uniform access interface to the highly diverse legacy systems without requiring special knowledge of internal, legacy interfaces of the linked systems.

[0028] Semantically, we model the back-end interface

as an XML document publishing and management system. We see the data source as "publishing or "serving" XML documents containing enterprise information. The back-end allows users to add, update, delete, browse, and search for documents in the data source. We chose this semantic model of interaction because it provides a generic interface through which many disparate legacy systems can be accessed.

[0029] A particular data source 111 can manage multiple types of documents, such as customer accounts, purchase orders, work items, work lists, and the like. Any document in any data source can be uniquely identified and retrieved by a document identification (id) 104. In our implementation, and keeping within the spirit of XML, we use a document identification 104 that is conceptually similar to a Web page Universal Resource Locator (URL), although different in detail. As shown, the service bridges include a bridge framework (BF) 113 and a data source-specific runtime access component (RAC) 114. The service bridge is described in greater detail below with reference to Figures 4-9.

Bridge Framework

[0030] The bridge framework 113 provides generic high level access services for the back-end interface. The framework is relatively independent from the specifics of the linked legacy systems and is implemented with reusable code. The bridge framework performs user authentication, and identifies the user making a request of the data source. The bridge framework also identifies agents 101 making requests, and provides a means to map a generic user identity to specific "logon" information required by any of the legacy data sources, e.g., a username and a password. The bridge framework operates securely such that any sensitive data-source logon information, such as a clear-text password, is encrypted.

Connection Pooling and Document Management

[0031] The framework also manages objects involved in establishing and maintaining a connection to the data source, and provides for connection sharing and pooling. Connection pooling and sharing is used when the establishment of a connection or session with the data source is too expensive to perform on a per user basis. The connection pooling and sharing mechanism is based on "user groups." All members of a user group access a particular data source via a shared connection pool. The connections in this pool are established within the user context of a "pseudo-user account."

[0032] A pseudo-user account is a special data source account that represents a group of users instead of an individual user. Thus, if we have two user names, "john@accounting" and "jim@accounting," the two accounting users both access the data source within the context of the accounting pseudo user account. Con-

nection pooling may not be necessary for all back-end data sources, but certainly is required for relational database access.

Document Caching

[0033] The bridge framework also provides a tunable caching facility to increase system performance. As stated above, a primary function of the back-end interface is to access legacy data and convert that data into the XML format. The bridge framework maintains XML documents in a cache 115 so that a subsequent request to retrieve the same data can bypass any data access or conversion work overhead by accessing the cached XML document.

[0034] The caching in our system is tunable. For a given type of document, a system administrator can specify caching parameters 116 such as whether caching should be enabled, a maximum lifetime before cache entries become stale, a maximum cache size, whether the cache 115 should be a persisted disk and re-used at next server startup. For document types that contain highly volatile data, caching can be disabled or cache entries can be set to expire quickly. For documents containing data that changes rarely, the caching parameters can be set aggressively to retain the documents in the cache.

Runtime Access Component

[0035] The runtime access component (RAC) 114 is specific for a particular data source 111. The RAC uses application programming interfaces (APIs) and structures of the legacy data source to access the data and to map the data into the XML format. The exact semantics of how the data are mapped to the XML format vary. For example, the mapping can be for widely used legacy databases, such as, JDBC, JDBT, SAP, or SQL. An example JDBC implementation is described below with reference to Figure 4. The RAC supports the following database access operations.

Query

[0036] The "query" operation retrieves a document from the data source. The caller supplies the *id* 104 of the document to fetch. The bridge service returns the specified information in the form of a XML document according to one of the standard programming models supported by W3C, for example, a DOM document object or a SAX document. DOM (Document Object Model), is a programming interface specification that specifies a tree which applications may then explore or modify. SAX is an event-based tool, more or less 'reading' the document to the application using a set of named methods to indicate document parts. SAX is typically used where efficiency and low overhead are paramount, while the DOM is used in cases where applications need

random access to a stable tree of elements. The interface allows us to generate and modify XML documents as full-fledged objects. Such documents are able to have their contents and data "hidden" within the object, helping us to ensure control over who can manipulate the document. Document objects can carry object-oriented procedures called methods.

[0037] In the case of a relational database, the query operation maps to a SQL SELECT statement with a WHERE clause specifying which record or records from the database are contain in the document.

Update

[0038] The "update" operation modifies existing data in the legacy data source. The caller supplies the *id* of the document and a XML document containing only the fields to be modified. In the case of the relational database, the update operation maps to a SQL UPDATE statement.

Delete

[0039] The "delete" operation removes a document from the data source. The caller supplies the *id* of the document to delete. In the case of the relational database, the delete operation maps to a SQL DELETE statement.

Add

[0040] The "add" operation inserts a new document into the data source. The caller supplies the document in the form of a DOM Document object. The bridge service returns the *id* of the newly added document. In the case of a relational database, the add operation maps to a SQL INSERT INTO statement.

Browse

[0041] The browse operation, also known as "buffering," browses all of the documents in the data source of a certain type. The caller supplies the type of document to browse. The bridge service returns a browse object similar to a JDBC result set. The browse object allows the caller to traverse the results in either direction, jumping to the first or last document, and to re-initiate the browse operation. In the case of a relational database, the browse operation maps to a SQL SELECT statement that returns multiple records.

Search

[0042] The search operation browses the data source for all documents of a certain type that meet a predefined search criteria. The search criteria can be a list of fields and values which the caller wants to match against records in the database. For example, the caller might

request all customer records that contain a "state" field matching the string "MA." The caller supplies the type of document to browse as well as a document containing the fields to be matched. The bridge service returns a browse object as above. In the case of a relational database, the search operation maps to a SQL SELECT statement in which the WHERE clause contains the LIKE operator.

Front-End Interface

[0043] The front-end interface 120 is responsible for user presentation and interaction. The front-end interface uses "forms" to allow users to view and modify information. As an advantage, the front-end interface provides a "thin" user interface, with simple interactivity that can easily be customized as the environment in the enterprise changes. The front-end forms use HTML 121, HTTP 122, Javascript, Java servlets 123, Java applets, and plug-ins as necessary. Being Web based, the user 103 can use any standard browser 124 to interact with the system from anywhere there is an Internet access point.

HTTP Communications

[0044] The HTTP is used as the communication mechanism between agents and users. The user 103 browses and modifies information, and initiates processes via the web browser 124. User requests are routed to agents 101 via HTTP and through the Java servlet. The servlet 123 in turn communicates with a front-end service bridge 125 that serves as an interface for the agents 101.

[0045] The servlet/service bridge combination 123/124 supports the establishment of user sessions that are the channel for two-way communication between the user and the agents. Within the context of a session, the user can send HTTP GET or POST requests to the agents, and the agents process such requests, and send back an HTTP response. Sessions allow the user to wait for an agent to arrive and allow an agent to wait for a user to connect.

HTML Form Style Sheets

[0046] We accomplish the display of information to users with HTML, web pages, and web forms. As stated above, the information that agents retrieve from data sources is in the form of the XML documents 102. To format the XML documents into a form suitable for users, the front-end servlet 123 converts the XML document to a HTML page using a style sheet 126, e.g. XSL, JSP or some other data replacement technique as described below. The result of this conversion is the HTML page containing the information in a user-friendly format. By applying the style sheet, the servlet recognizes and replaces certain data from the XML document and

converts the data to HTML form.

[0047] For example, a particular XML document 102 includes the following information:

```
<customer>:
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</customer>
```

[0048] The HTML style sheet 126 for this document is as follows:

```
<html>
  <h1>'customer.firstname'</h1>
  <h2>'customer.lastname'</h2>
</html>
```

[0049] After applying the style sheet to the XML document, the resultant HTML form 121 would appear as:

```
<html>
  <h1>John</h1>
  <h2>Smith</h2>
</html>
```

[0050] The style sheet supports accessing all of the elements and attributes in the XML documents, and iteration over groups of repeating elements.

[0051] For example, an XML document contains:

```
<customer type="preferred">
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</customer>
```

[0052] The "type" attribute of the customer is accessed by using a syntax such as the following:

```
'customer.attr[type]'
```

which yields the value "preferred." Given a document containing repeating groups as follows:

```
<customers>
  <customer type="preferred">
    <lastname>Smith</lastname> </customer>
  <customer type="standard">
    <lastname>Jones</lastname>
  </customer>
```

[0053] The "lastname" element of the second customer is accessed using a syntax such as 'customer[1].lastname' which yields the value "Jones." To iterate over all of the customers and access their "type" attributes, an expression such as:

```
'iterate(i=customers.customer) {
  i.attr[type]
```

can be used to produce first the string "preferred," and then "standard."

Validation

[0054] The front-end interface also supports the validation of user entered information. Field validation information supplies some immediate feedback and interactivity to the user. Field validation also increases application efficiency by detecting common errors within the web browser process before any other network traffic is incurred or application logic is executed. Client side validation can be broken down into two related levels.

BEST AVAILABLE COPY

Field-Level

[0055] Field-level validation performs simple checks on user entered data to validate that the information is of the correct format or data type. For example, field-level validation can validate that a user enters numeric values in a particular field, or uses a proper date format. We implement field-level validations with Javascript. A library of common validations is supplied as a script file on a web server. The library has a ".js" file extension. This script file can be included into HTML forms as desired using the <script> HTML tag. Validation is enabled for a field by indicating the name of an appropriate validation routine, e.g. "onChange," within an event handler of the field. The event handler is triggered when an INPUT field changes. Setting up validation for a field requires HTML coding as follows:

```
<input type="text" name="birthdate" onChange="validateDate(birthdate)">
```

The validation library provides routines for common data types such as dates, times, currency, etc. The validation library can also provide a pattern matching ability allowing user input to be matched against arbitrary patterns, e.g., a pattern \$##.## to match a monetary amount.

Cross-Field Validation

[0056] Cross-field validation allows for more complex validations. In this type of validation, the contents of one field depends on the contents of another field. For example, cross-field validation can detect a situation where a telephone number must be entered. Such validation usually requires a more detailed knowledge of the requirements of the application.

Middle Tier

[0057] The middle tier 130 provides the "glue" that links the back-end and the front-end interfaces. The middle tier utilizes the mobile agents 101 to communicate with the interfaces. The middle tier also provides support for disconnected applications and users. In addition, the middle tier customizes the system 100 to the needs of specific enterprise functions without actually having to reprogram the legacy systems.

[0058] The middle tier supports the automation of complex workflow and complex validations of data that may require access to multiple data sources. As a feature, the middle tier uses a rules engine (RE) 131 operating on rules stored in a database 132. The rules are defined in a rules language, and can be retrieved by the agents 101 as needed.

[0059] In a typical scenario, the user launches an agent 105 due to interaction with the browser 124. The agent carries an XML document, e.g., a purchase order 106, to the rules database 132. The agent retrieves the appropriate rule for processing the order, such as a purchase order workflow. The agent then interprets the rule

to appropriately route the document to the locations in the network specified by the rule. The rule can include a travel itinerary, as well as instructions on how to interact with the data sources.

5 [0060] As an advantage, the operation of our system is always current. As rules change so does the operation of the system. The agents always execute according to the current state of the rules database.

10 Design Tools

[0061] As shown in Figure 2, the primary purpose of the design tools 140 is to generate 141 XML document type definitions (DTD) 142, to specify 143 data mappings, i.e., RACs 114, to encode 144 rules 132, and to design 145 user interfaces 126.

Document Type Definitions

20 [0062] The step 141 identifies the different types of document information (DTD) 142 that needs to be shared by the various data sources 111 of the back-end 110 and the browser 124 of the front-end 120. This information is specified in the DTDs. For example, to share purchase order information between systems, the type of information needed in a purchase order needs to be identified, then that information needs to be encoded in a corresponding DTD. In one embodiment, the design tools use the service bridge to extract schemas from the data sources.

Data Mapping

35 [0063] After a data source independent data format has been generated, the mappings between the XML format and legacy formats for a particular database needs to be specified as shown in Figure 3. A query operation to a relational databases 111 involves extracting the schema of the database by generating a SQL runtime access component (RAC) 114 which makes the JDBC calls to the database, converting the resulting data into the XML format, and handing the XML document 113 to an agent 101. The access components can be implemented as Java code. The agent delivers the XML to the front-end 120 for conversion to the HTML form 121 using the style sheet 126 so that the data can be viewed by the user 103 using a standard browser 124.

45 [0064] Conversely, the update operation converts the HTML form to the corresponding XML document. The XML document is converted to a legacy format and the RAC modifies the data source using its schema. For other legacy data sources that are not specified by a schema or some other metadata, the mapping may need to be done by means that access the APIs directly.

Rule Encoding

55 [0065] After the data format definition is generated,

and the RAC has been specified to access the appropriate data source, the next step is to encode what agents are going to do with the information. In a simple data replication system, an agent may retrieve modified records from a master database, travel to the location of a backup database, and then update the backup database with a copy of the modified record. This process involves the encoding of a specific rule.

Designing the User Interface

[0066] As shown in Figure 2, generating the user interface requires three steps: manipulating document type definitions (DTD) 145, importing DTD 146, and generating DTD from database schema 147.

Authoring DTD

[0067] The design tools 140 allow the system designer to define, design, and manipulate XML and HTML DTDs. A DTD 142 defines the name of the following document elements: the contents model of each element, how often and in which order elements can appear, if start or end tags can be omitted, the possible presence of attributes and their default values, and the names of the entities.

[0068] Because the DTDs represent many different types of documents in the system, this step essentially defines the data types of the enterprise's computerized applications. As an advantage, the resulting DTDs do not directly tie the system to any specific legacy data source, nor do the definitions preclude the integration of other legacy systems in the future.

DTD Import

[0069] The tools also allow one to import already existing DTD definitions. Such functionality can be used in environments where DTDs have already been defined for standard document types. These DTDs may have been defined by standards bodies or a designer of the legacy system.

DTD generation from Database Schema

[0070] This part of the tools automatically generate DTDs from existing database schema.

XML ↔ SQL Mapping Definition

[0071] Given the existence of the DTDs, the system 100 provides tools that map between legacy back-end data formats and XML document formats. In the case of relational database access, these mappings link tables, columns, and fields from the legacy database to elements and attributes of the XML documents as defined by the DTDs. This also allows the definition of several distinct mappings, each of which involves accessing

slightly different information in the data source.

Data Mappings

5 Query Mapping

[0072] A query mapping enables an agent to retrieve information from a legacy data source. In the case of a relational database, this mapping specifies the contents of the SELECT statement, including any information relevant for a table join. A query mapping for a purchase order may involve accessing a purchase order table, a customer table, and a product catalog table.

15 Update Mapping

[0073] An update mapping allows an agent to modify information in the data source. This involves specifying the contents of an UPDATE statement. An update mapping for a purchase order involves updating the purchase order table, but not modifying the customer table or the product catalog table.

Delete Mapping

[0074] A delete mapping allows an agent to delete information in the data source. This involves specifying the contents of a DELETE statement. A delete mapping for a purchase order involves deleting a record or records from the purchase order table, but not modifying the customer table or the product catalog table.

Add/Create Mapping

[0075] An add/create mapping allows an agent to add information to the data source. This involves specifying the contents of an INSERT statement. An insert mapping for a purchase order involves adding a record or records to the purchase order table, but not modifying the customer table or the product catalog table.

Schema Extraction and Caching

[0076] In order to allow for mapping between a legacy database schema and XML DTD formats, the mapping design tool extracts the schema from legacy databases. Because schema extraction is an expensive and time consuming task, the tools allow one to save extracted schemas on a disk for subsequent use.

Form Generation

[0077] The tools will also allow one to automatically generate a form from a DTD. Such a form may require minor modifications to enhance the physical appearance of the form. For example, color or font size of text can be adjusted to enhance usability.

Embedding Binary Data in XML Documents

[0078] Some enterprise applications may need to retrieve arbitrary binary data from the data source 111. For example, a legacy database contains employee information. Included with that information is a picture of the employee in standard JPEG format. The employee information is stored as a single table named "employees," which has a schema as Table 1, where the field <image> represents the picture:

Table 1

ID	Name	HireDate	Photo
1	John Smith	1/1/96	<image>

[0079] The XML document that retrieves the above table appears as follows:

```
<employee>
  <ID>1</ID>
  <name>John Smith</name>
  <hiredate>1996-29</hiredate>
</employee>
```

XML, by itself, does not naturally lend itself to the inclusion of binary data. To deliver this information for display in a web page, the service bridge 112 could encode the SQL record in an XML document as follows:

```
<employee>
  <ID>1</ID>
  <name>John Smith</name>
  <hiredate>1996-29</hiredate>
  <Photo href="http://server/directory/john.
jpeg"/>
</employee>
```

[0080] However, there are a number of problems with this type of approach. First, it is the responsibility of the user to issue the proper additional commands to retrieve the linked document before it can be displayed, e.g., the user must click on the URL of the picture. Second, the DTD for the XML document must specify the URL. For most legacy databases, it is unlikely that the records storing the binary data are accessible via an HTTP URL. Furthermore, the binary data is transported through the system by a follow on transport, such as HTTP. For reliability, security, consistence, and other reasons we prefer to carry all data, including binary data with the agents.

[0081] To allow the servlet 123 to generate an agent that can access the binary data, we define a new type of URL. The new URL incorporates the location of the binary data, as well as a unique "name" that can be used to retrieve the binary data. The URL contains the host-name of the data source, a service name, an action name that can be used to perform the retrieval of the binary data, and a document identification referring to the binary data. This still results in a fairly complex URL.

[0082] Using multiple requests to retrieve the binary data is inconsistent with our agent model. Agents try to

use the network effectively by batching data into fairly large self-contained packets. This is very different than the hypertext model used on the web in which a single page display can lead to multiple network requests.

Compound Documents

[0083] In an alternative solution, we define a compound document. In a compound document, the binary data is embedded in the same document as the textual XML data. This approach is consistent with our agent driven system that attempts to transport data as larger batches. Compound documents can be built in two ways.

Embed Binary Data Into XML Text Element

[0084] The binary data is embedded directly into an XML text element. This can be done as long as the binary data is encoded in such a way that the data only contain XML characters. Such an encoding could be based on the Base64 encoding. With Base64, special characters, such as "<" and ">," are replaced with equivalent entities (i.e., < and >). We also can use a character data (CDATA) section to work around the problem of illegal characters within the Base64-encoded data. We may want to prefix the embedded binary data with standard mime headers that specify content type, encoding, and name. Such a format for the photo element appears as follows:

```
<Photo>
  Content-Type: image/jpeg
  Content-Encoding: base64
  Content-Name: john.jpeg
  9/4AAQSkZJ....gEASABIAAD/
</Photo>
```

[0085] It should be noted that this alternative increases the size of the binary data by 33% as well as increasing the overhead to encode and decode the data.

[0086] This alternative requires that a SQL RAC extracts the binary data and encodes the data into Base64, and then adds the encoded data to the XML document with the proper mime headers.

Compound Document Encoded as Mime Document

[0087] Another alternative, embeds both the XML document and the binary data into separate parts of a multipart mime document. Each part of the overall document has a Content-ID which is referenced from a standard XML link, in part, such a format appears as follows:

```
Content-Type: multipart/related; boundary= "--XXXXX"
--XXXXX
Content-Type: text/xml
Content-ID: doc
  <Photo href="cid:photo"/>
--XXXXX
```

Content-Type: image/jpeg
 Content-Encoding: base64
 Content-Name: john.jpeg
 Content-ID: photo
 9j/4AAQSkZJ... gEASABIAAD/
 ----XXXX----

[0088] With this alternative, the binary data may not need to be encoded. However, this requires that agents also retrieve MIME documents via the RAC.

JDBC Service Bridge

[0089] Figure 4 shows details of a preferred embodiment of a service bridge 400 of the back-end interface 110 for accessing a data source. In this embodiment, JDBC is used to access a SQL type of database. The bridge 400 includes a public interface 410, JDBC run-time access component (RAC) 420, XML-SQL data mapping 430, and a document cache 440 as its main components.

Public Interface

[0090] As stated above, the public interface 410 provides the means by which agents access the data sources 111. The public interface allows data retrieval, modification, and addition. As an advantage, the public interface 410 makes no assumptions about how data in the legacy database 111 is sourced or maintained. Instead, we make the public interface resemble the GET/PUT model of HTTP.

JDBC Run-Time Access Component

[0091] The JDBC access component 420 is responsible for establishing and managing JDBC connections, building and executing SQL statements, and traversing result sets. This component works entirely within the context of JDBC and SQL.

XML-SQL Data Mapping

[0092] The XML-SQL data mapping 430 uses the mapping information generated by the design tools 140 to map data between XML and SQL.

Document Cache

[0093] The document cache 440 operates entirely with XML documents. XML documents that have been retrieved from the data source can be cached for fast future retrieval. The caching services are configurable so that maximum cache sizes and cache item expiration times can be specified. Caching can be disabled for certain classes of documents which contain highly volatile information.

[0094] Figure 5 shows the public interface 410 in greater detail. The interface supports four basic types

of accesses, namely get 510, put 520, add 530, and delete 540.

[0095] At the heart of the interface is the document *id* 104. The document *id* is a string which uniquely identifies every document instance within the data source. The document *id* can be thought of as corresponding to the URL of a World Wide Web document, or to the primary key of a record in a database. Although the *id* has a different format than a URL, it does serve as a document locator.

[0096] In order to interact with information in the legacy data source, an agent needs to provide the *id* for the document containing the information. The *id* contains multiple sections of information and follows the following pattern.

[0097] The first character of the *id* string specifies a separator character (S) 501 that is used to separate the different sections that make up the document *id*, e.g., a colon (:). This character is used in conjunction with a Java StringTokenizer to parse the document *id*. The subsequent information in the *id* includes name=value pairs (N, V) 502. One pairs 502 specifies a document type, e.g., ":type=cust_list:"

[0098] In most common cases, the *id* 104 also contains a key specifying the exact document instance in order to uniquely identify an individual document in a data source. For example, in a document containing customer information, this key contains a data source specific customer number or a customer *id*. Within the service bridge, this key is mapped to a WHERE clause of a SQL statement. For example, an agent can request customer information for a particular customer by specifying an *id* string as follows:

" :type=customer:key=SMITH:"

This request results in a SQL query to the database that appears as follows:

SELECT * FROM Customers WHERE Customers.ID=SMITH

The exact semantics of how they key is mapped into the resultant SQL statement is specified by the design tools 140.

[0099] The key portion of the *id* can be composed of multiple pieces of information separated by, for example, commas. Such a key is used in cases in which the WHERE clause of the corresponding SQL query needs multiple pieces of information to be specified by the agent. An example of this is a document containing a list of customers, where the customers names are within a certain alphabetic range, for example, "all customers whose last names begin with the letters A or B. Such a document has an *id* as follows:

" :type=cust_list_by_name:key=A,Bzzzz:"

In this case, the request would map into a SQL statement resembling the following:

SELECT * FROM Customers
 WHERE Customers.LastName BETWEEN A, Bz-

zzz

Implementation Details of the Service Bridge

Database Access

User Authentication

[0100] The service bridge is responsible for performing any authentication necessary in order to establish a database connection. This may involve supplying a database specific username and password or other login information. When a database access (get, put, add, delete) is made by an agent, the bridge examines the agent's runtime context to determine the user identity associated with the agent.

[0101] After the agent's identity has been ascertained, the service bridge maps the identity into simultaneous database-specific user identification using a mapping table generated by the design tools. For example, the mapping maps the user identity "steve@accounting" into an Oracle username "steve."

[0102] In order to establish a connection to a database on behalf of a user, the service bridge retrieves both the username and clear-text password for the corresponding database user account. In such cases, the clear-text password is stored in the identity-mapping table. For security reasons, the table is encrypted on disk using a public/private key pair.

Connection Management

[0103] To enhance performance and scalability, the service bridge supports database connection pools. This means that multiple users share a common pool of JDBC connections. Establishing a database connection can be a slow and relatively expensive operation. The use of shared connection pools decreases this expense.

[0104] The basis for this connection sharing are "users groups." When an agent attempts an operation which requires a connection to a database, the service bridge performs that operation using a connection established in the context of a special "pseudo-user" account. The pseudo-user is a database system account that represents not an individual user, but instead a particular group of users. A pool of such pseudo-user connections is available for use by all of the agents of the group. The service bridge generates and maintains a connection pool for each distinct group of users who access the bridge.

[0105] Figure 6 shows agents 101 for three users *tom*, *joe* and *david* 601-603 accessing the data source 111. Two of the users, *tom@users* and *joe@users*, are members of a users group. The third user, *david@managers*, is a member of a "managers" group. When these agents attempt to access the database, the two members of the users group share a connection pool 610 that was established with the credentials of the "users" pseudo-user. The third agent will communicate with the database using a separate connection pool 620 established with

the credentials of the "managers" pseudo-user.

[0106] A connection pool for a particular group is generated when a member of the group makes the first access request. Connections within the pool are constructed as needed. The service bridge does not pre-allocate connections. After a configurable, and perhaps long period of inactivity, the connection pool is closed to free database resources. If a connection pool for a particular group has been closed due to inactivity, then any subsequent request by a member of that group results in the generation of a new pool. When a request is completed, the connection allocated for that request is returned to the pool. A maximum number of connections in a pool can be specified. If no connections are available when a request is made, then the request is blocked until a connection becomes available.

Statement Construction and Execution

[0107] The actual generation and execution of SQL statements is performed by a separate "modeler" object. The modeler object is generated by the design tools 140. For each type of document used in the system, there is a distinct modeler object. Each modeler knows how to construct exactly one type of document. During the design process, one specifies what information is to be retrieved from the database, and how to map the information into an XML document. The design tools serialize and save the modeler objects in a ".ser" file. At runtime, the service bridge loads and de-serializes the modeler objects from the ".ser" file. The resultant modeler objects are able to perform all of the data access and mapping functions required to retrieve information from the data sources. As stated above, SQL to XML data mapping is performed by the modeler object designed for a particular document type.

Data Caching

[0108] To improve the performance of document retrieval, the data service caches database information as converted XML documents. When a first request is made to retrieve a document, the service performs the SQL access and SQL to XML data mapping as described above. The resultant XML document is added to the cache of documents 440 maintained by the service bridge. Any subsequent request to retrieve the document will be satisfied by retrieving the document from the cache, bypassing the need for an additional expensive database access and mapping.

[0109] When an update or addition is made to a data source, the cache is updated to reflect the new information. The update to the cache is made only after the SQL statement performing the update of the end database has been completed successfully. This prevents the cache from storing information that has not been committed to the database due to errors or to security restrictions.

[0110] The XML document cache is configurable to specify a maximum size of the cache, the maximum amount of time a single document can be retained in the cache before it becomes stale, and whether the cache should be persisted to disk, in which case the cache can be re-used after a server restart. One can also customize how different classes of documents are cached. If a document represents highly volatile information, then caching can be disabled for that class of document. If a document class is completely (or virtually) static, then documents of that class can be cached for a very long time.

Execution Flow

[0111] The following section describes the execution flow for basic database access requests. Figure 7 shows the steps 700 of a "get" or retrieval access in greater detail. After the request is received from the agent 710, the caller and document identity are determined 720, 730. The group specific cache is identified 740, and the cache is checked 750. If the cache stores the document, return the document in step 755. Otherwise, locate the XML-SQL mapping 760, construct the select SQL select statement 770, retrieve the connection 775, and execute the statement in step 780. Next, the result set is "walked" 785, fields are extracted 790 to build the XML document 794, the document is cached 796 and returned to the agent in step 798. Figure 8 shows the steps 800 for the addition (add) and modification (put) similar to the get steps. The delete request simply deletes data from the database as shown at 540 in Figure 5.

Run-time Object Hierarchy

[0112] Figure 9 shows the run-time hierarchy 900 of objects of the service bridge 110. The objects can be classified as data source independent 901, and data source dependent 902. The data source independent object 901 includes data source factory object 910 indexed by group name, group specific data source objects 920, document factory objects 930 (one per document), document cache objects 940, document builder objects 950, connection pool objects 960, mapping table objects 970, document manager objects 980, and the data source manager objects 990. The data source dependent object 902 include source connection 991, string authentication 992, document map 993, and specific driver objects 994.

[0113] Although the invention has been described by way of examples of preferred embodiments, it is to be understood that various other adaptations and modifications may be made within the spirit and scope of the invention. Therefore, it is the object of the appended claims to cover all such variations and modifications as come within the true spirit and scope of the invention.

Claims

1. An enterprise integration system coupled to a plurality of data sources, the plurality of data sources using different data formats and different access methods, comprising:

a back-end interface configured to convert input data source information to input XML documents and to convert output XML documents to output data source information;
a front-end interface including means for converting the output XML documents to output HTML forms and for converting input HTML forms to the XML documents;
a middle tier including a rules engine and a rules database;
design tools for defining the conversion and the XML documents;
a network coupling the back-end interface, the front-end interface, the middle tier, the design tools, and the data sources;
a plurality of agents configured to communicate the XML documents over the network and to process the XML documents according to the rules.

2. The system of claim 1 wherein the back-end interface further comprises:

a public interface;
a document cache; and
a run-time access component.

3. The system of claim 2 wherein the public interface forwards the input XML document to the plurality of the agents for distribution, and the public receives the output XML documents for storing in the plurality of data sources.

4. The system of claim 2 wherein the document cache includes caching parameters.

5. The system of claim 2 wherein the caching parameters include a maximum lifetime for each cache entries, a maximum cache size, and a persistency indicator.

6. The system of claim 2 wherein the run-time access component generates access requests for the plurality of data sources.

7. The system of claim 6 wherein the access requests include query, update, delete, add, browse, and search.

8. The system of claim 1 wherein the XML documents include binary data.

9. The system of claim 8 wherein the binary data is referenced by a Universal Resource Locator.
10. The system of claim 8 wherein the binary data is embedded as a compound document. 5
11. The system of claim 10 wherein the compound document embeds the binary data as an encoding in a character set. 10
12. The system of claim 10 wherein the compound document embeds the binary as a MIME document.
13. The system of claim 1 wherein the input documents are presented to a browser. 15
14. The system of claim 1 wherein each XML document is identified by a document identification.
15. The system of claim 14 wherein the document identification is a character string. 20
16. The system of claim 15 wherein the character string includes a plurality of sections, and a first character of the string is a section separator. 25
17. The system of claim 16 wherein one of the sections stores a document type.
18. The system of claim 15 wherein one of the sections stores a key to an instance of the XML document in one of the data sources. 30
19. The system of claim 1 wherein the back-end interface performs user authentication. 35
20. The system of claim 1 wherein the back-end interface supports database connection pools.
21. A method for integrating a plurality of data sources, the plurality of data sources using different data formats and different access methods, comprising: 40
 - converting input data source information to input XML documents and converting output XML documents to output data source information; 45
 - converting the input XML documents to input HTML forms and converting output HTML forms to the output XML documents; 50
 - providing a rules engine and a rules database; defining the converting and the XML documents;
 - communicating the XML documents over a network using agents; and 55
 - processing the XML documents by the agents according to the rules database.

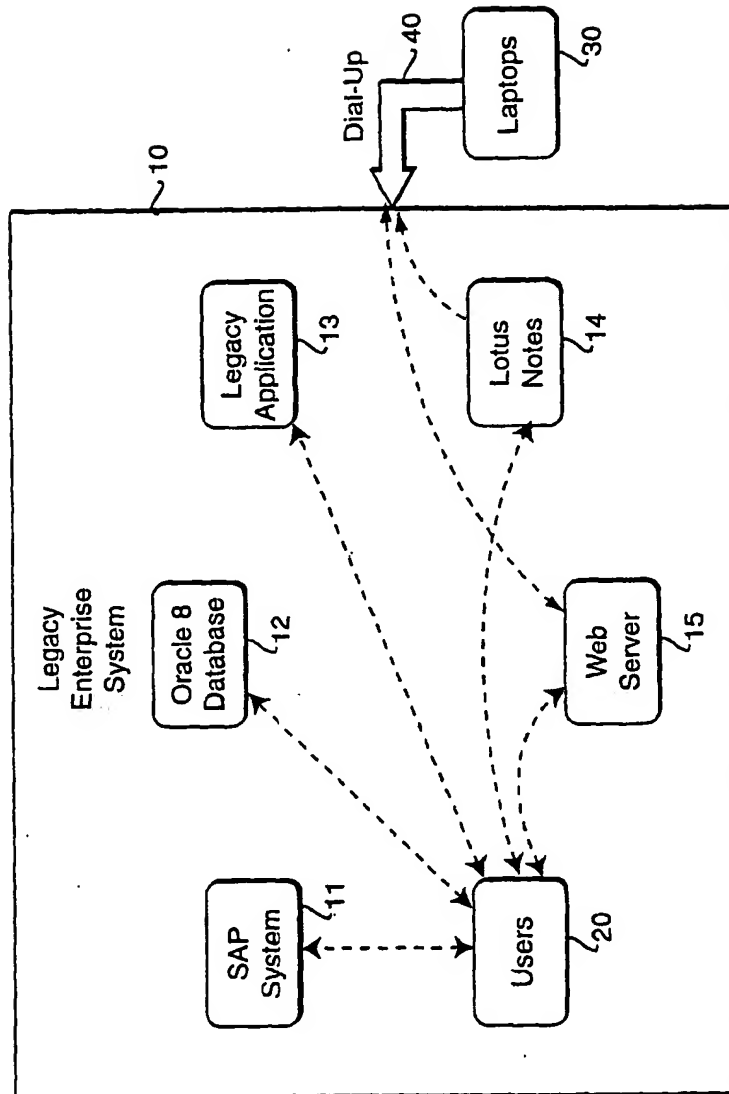


FIG. 1a
PRIOR ART

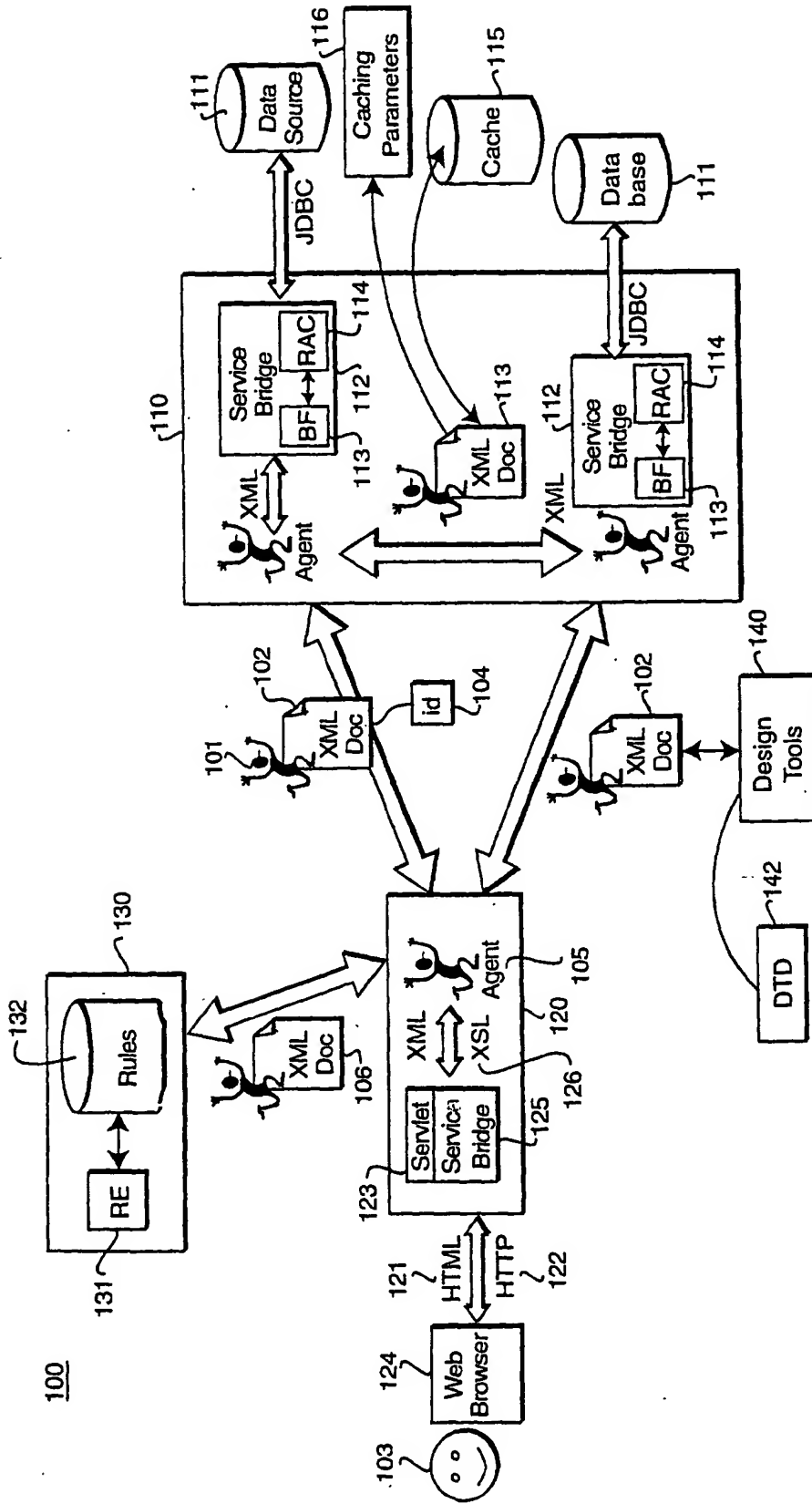


FIG. 1b

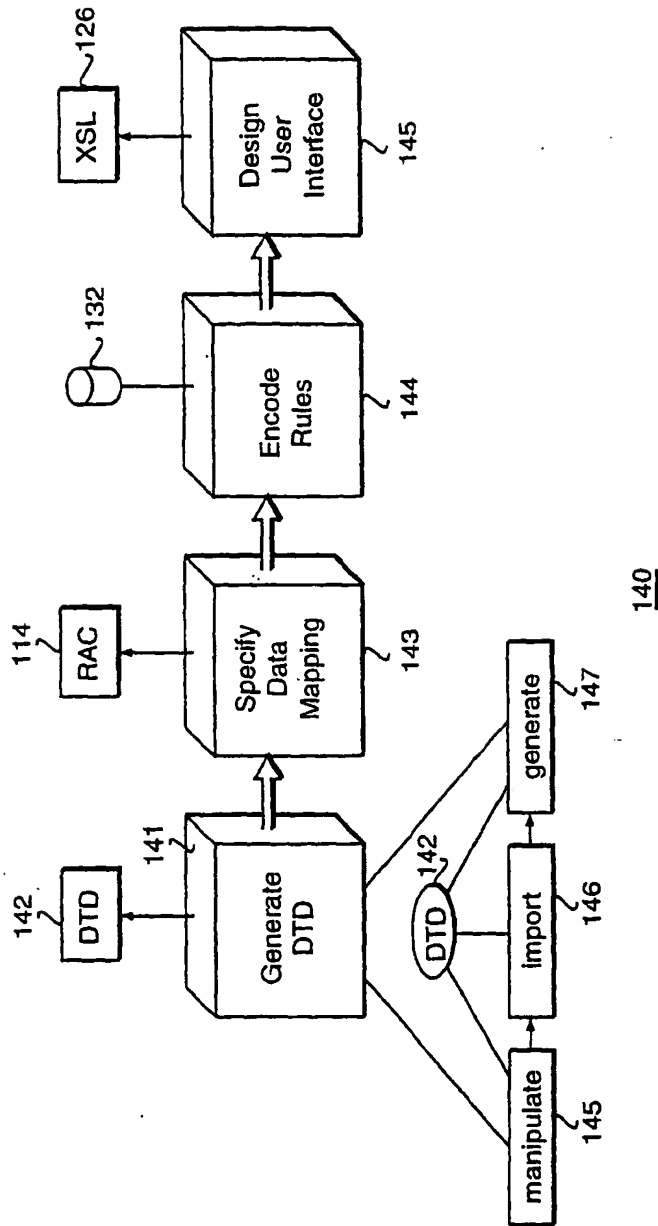
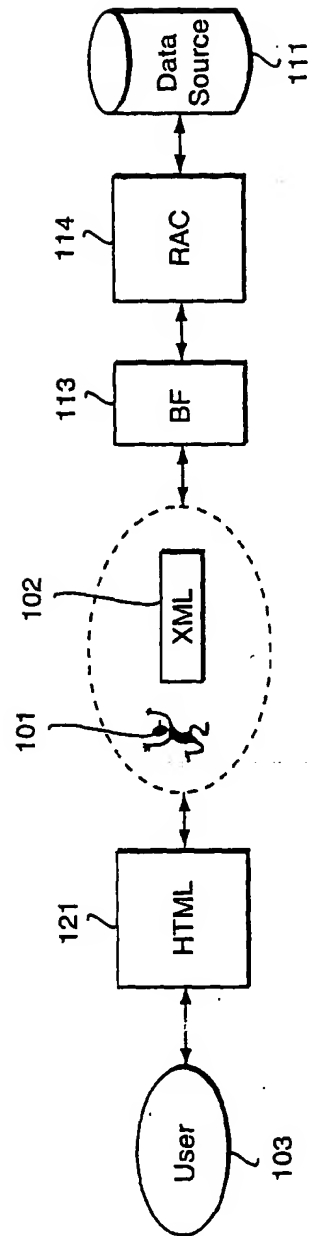


FIG. 2



300

FIG. 3

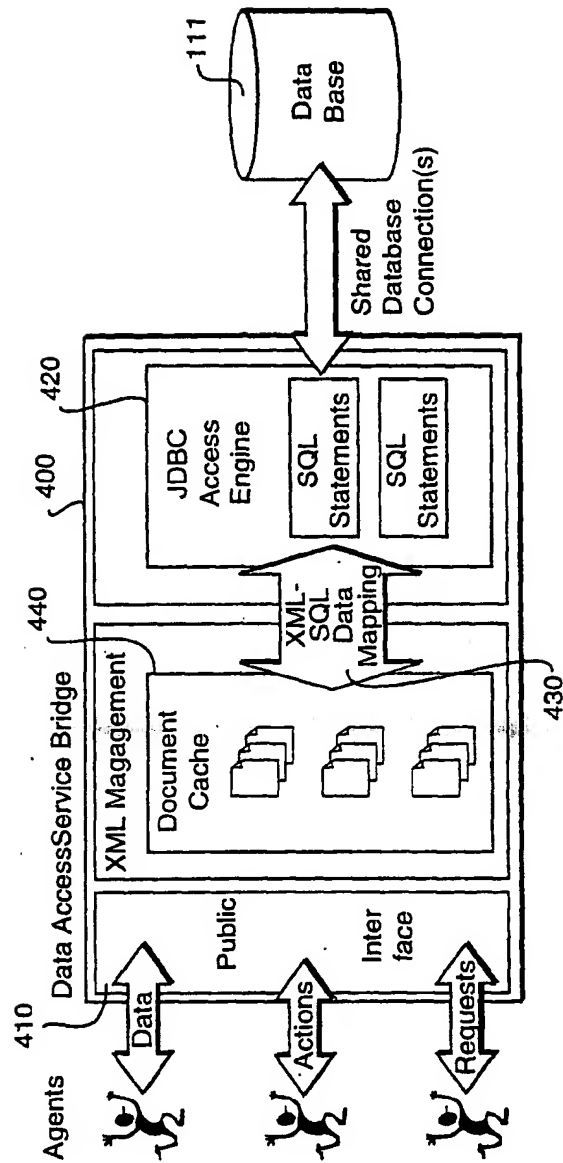
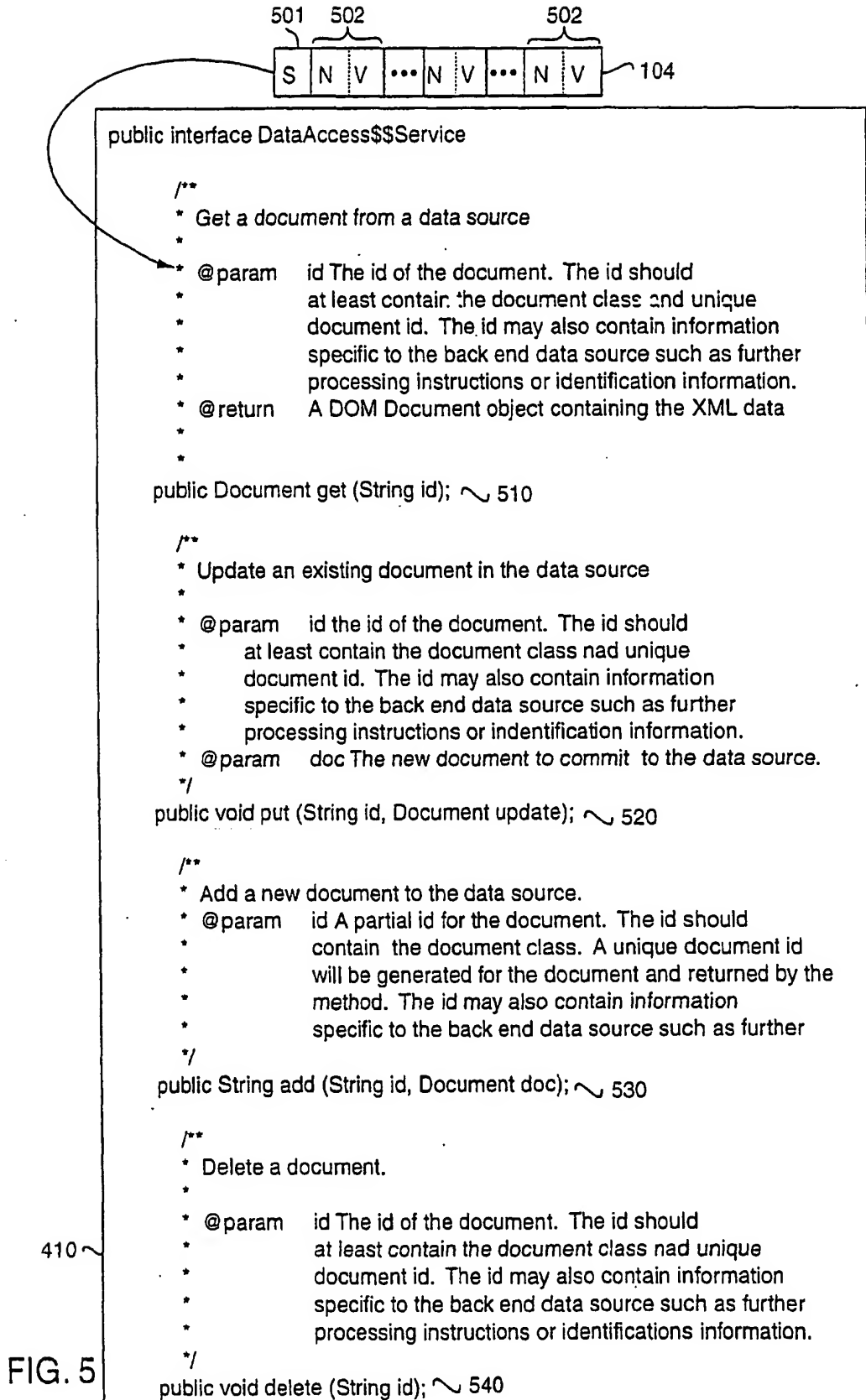


FIG. 4



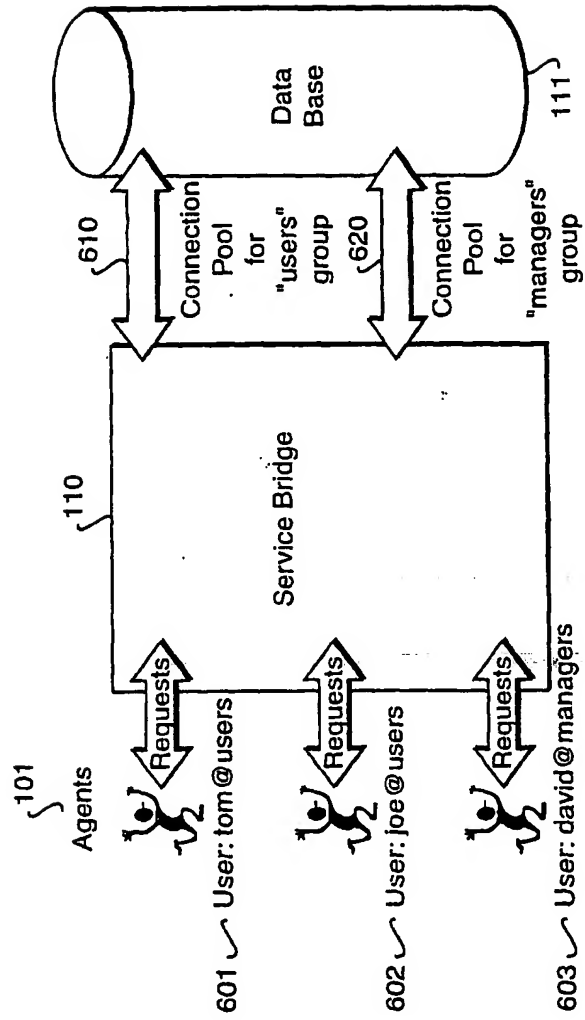


FIG. 6

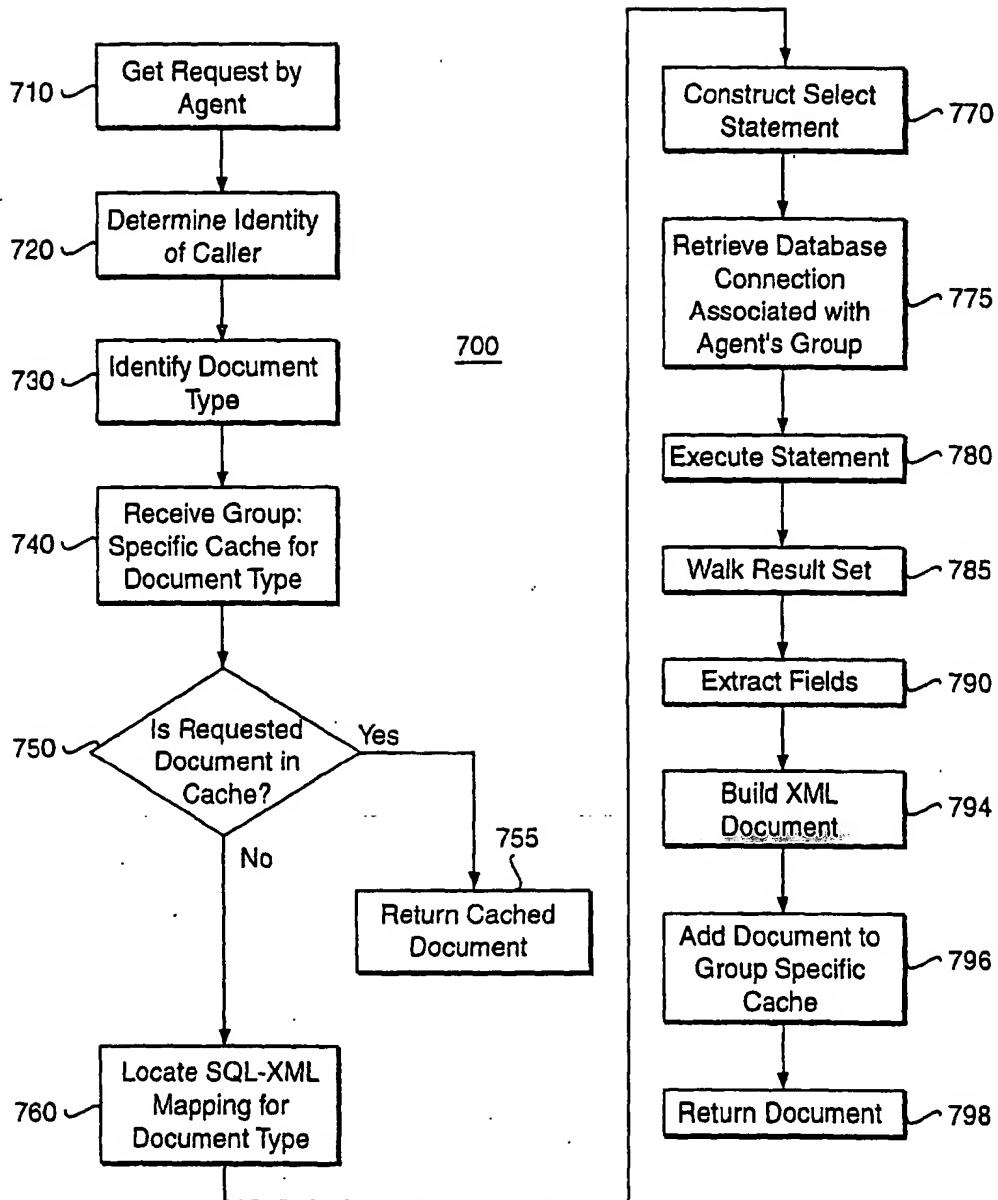


FIG. 7

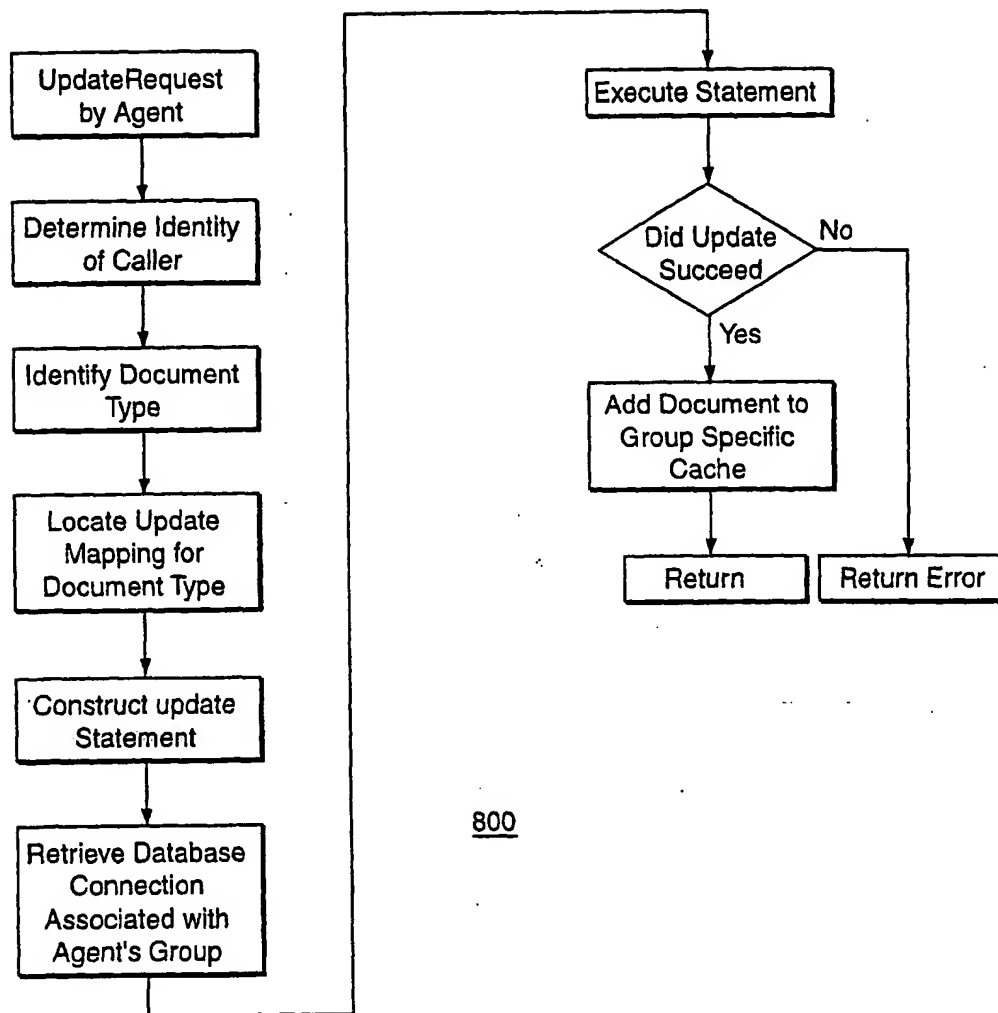
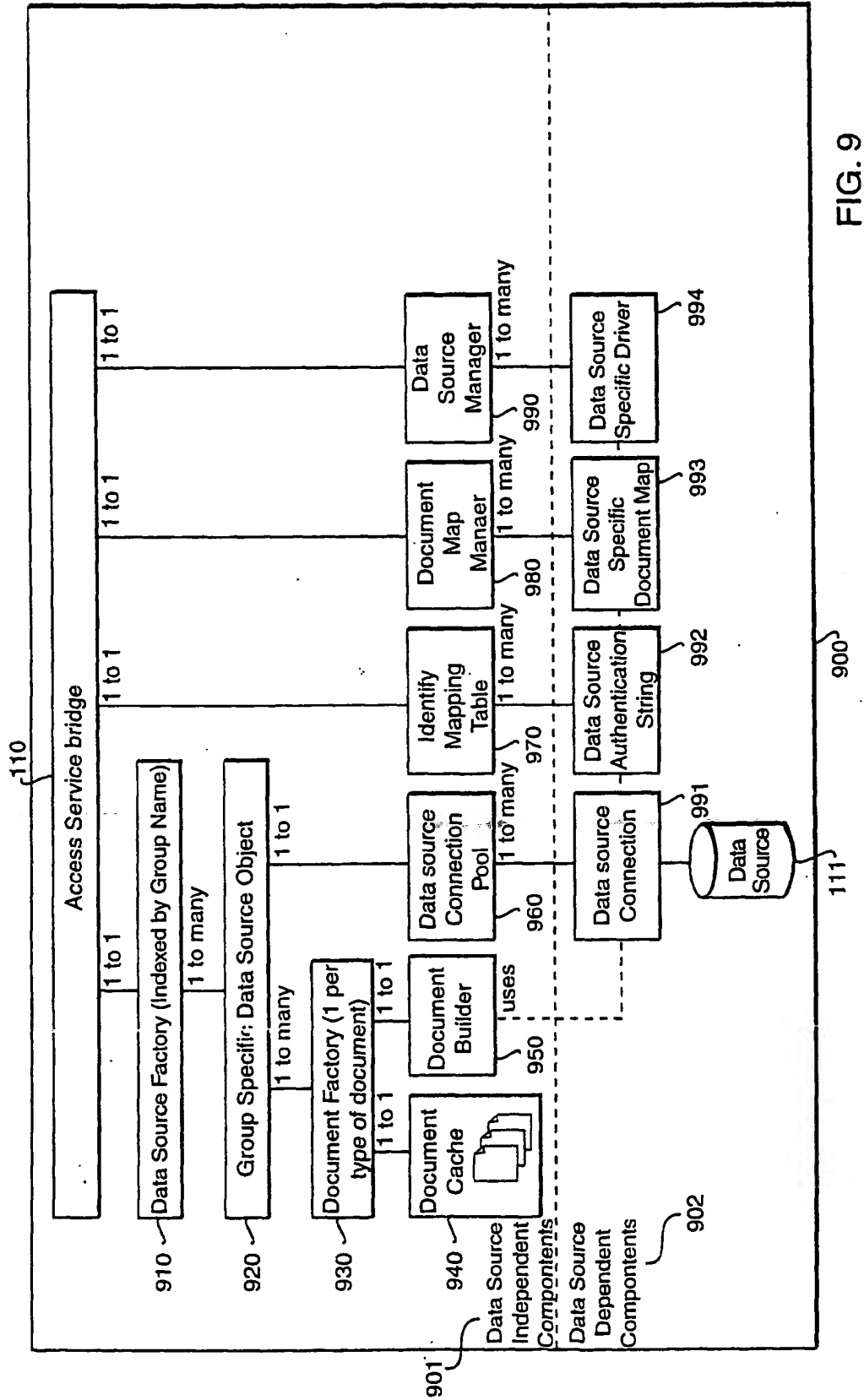
800

FIG. 8





European Patent
Office

EUROPEAN SEARCH REPORT

Application Number
EP 01 10 2032

DOCUMENTS CONSIDERED TO BE RELEVANT			
Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.7)
P,X	EP 1 030 254 A (SD & M) 23 August 2000 (2000-08-23) * abstract; figure 1 *	21	G06F17/30
A	--- "Microsoft Announces Finalized BizTalk Framework" MICROSOFT PRESSPASS, [Online] 6 December 1999 (1999-12-06), XP002166805 Redmont, Wash. Retrieved from the Internet: <URL:http://www.microsoft.com/PressPass/press/1999/Dec99/BIZTALK1.0pr.asp> [retrieved on 2001-05-08] * the whole document *	1,21	
A	--- VAN VOOREN L: "XML and legacy data conversion: introducing "consumable documents" SGML EUROPE '97. CONFERENCE PROCEEDINGS, PROCEEDINGS OF SGML '97. THE NEXT DECADE - PUSHING THE ENVELOPE, BARCELONA, SPAIN, 13-15 MAY 1997, [Online] pages 185-187, XP002166806 1996, Alexandria, VA, USA, Graphic Commun. Assoc, USA Retrieved from the Internet: <URL:http://www.infoloom.com/gcaconfs/WEB/barcelona97/vooren57.HTM> [retrieved on 2001-05-08] * the whole document *	1,21	TECHNICAL FIELDS SEARCHED (Int.Cl.7) G06F
A	--- WO 99 23584 A (FRIEDMAN MARK M ;IOTA IND LTD (IL); STERN YONATAN PESACH (IL)) 14 May 1999 (1999-05-14) * abstract * * page 2, line 21 - page 4, line 21 *	1,21	
The present search report has been drawn up for all claims			
Place of search BERLIN		Date of completion of the search 8 May 2001	Examiner Triest, J
<p>CATEGORY OF CITED DOCUMENTS</p> <p>X : particularly relevant if taken alone Y : particularly relevant if combined with another document of the same category A : technological background O : non-written disclosure P : intermediate document</p> <p>T : theory or principle underlying the invention E : earlier patent document, but published on, or after the filing date O : document cited in the application L : document cited for other reasons & : member of the same patent family, corresponding document</p>			

EPO FORM 1503 03/82 (P04C01)

**ANNEX TO THE EUROPEAN SEARCH REPORT
ON EUROPEAN PATENT APPLICATION NO.**

EP 01 10 2032

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report.
The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

08-05-2001

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 1030254 A	23-08-2000	NONE	
WO 9923584 A	14-05-1999	US 6161107 A	12-12-2000
		AU 1371599 A	24-05-1999

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82